

AUTOMATED TELESCEINCE: ACTIVE MACHINE LEARNING OF REMOTE DYNAMICAL SYSTEMS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Daniel Le Ly

August 2013

© 2013 Daniel Le Ly
ALL RIGHTS RESERVED

AUTOMATED TELESCEINCE: ACTIVE MACHINE LEARNING OF REMOTE DYNAMICAL SYSTEMS

Daniel Le Ly, Ph.D.

Cornell University 2013

Automated science is an emerging field of research and technology that aims to extend the role of computers in science from a tool that stores and analyzes data to one that generates hypotheses and designs experiments. Despite the tremendous discoveries and advancements brought forth by the scientific method, it is a process that is fundamentally driven by human insight and ingenuity. Automated science aims to develop algorithms, protocols and design philosophies that are capable of automating the scientific process. This work presents advances the field of automated science and the specific contributions of this work fall into three categories: coevolutionary search methods and applications, inferring the underlying structure of dynamical systems, and remote controlled automated science.

First, a collection of coevolutionary search methods and applications are presented. These approaches include: a method to reduce the computational overhead of evolutionary algorithms via trainer selection strategies in a rank predictor framework, an approach for optimal experiment design for nonparametric models using Shannon information, and an application of coevolutionary algorithms to infer kinematic poses from RGBD images.

Second, three algorithms are presented that infer the underlying structure of dynamical systems: a method to infer discrete-continuous hybrid dynamical systems from unlabeled data, an approach to discovering ordinary differential equations of arbitrary order, and a principle to uncover the existence and dynamics of hidden state variables

that correspond to physical quantities from nonlinear differential equations. All of these algorithms are able to uncover structure in an unsupervised manner without any prior domain knowledge.

Third, a remote controlled, distributed system is demonstrated to autonomously generate scientific models by perturbing and observing a system in an intelligent fashion. By automating the components of physical experimentation, scientific modeling and experimental design, models of luminescent chemical reactions and multi-compartmental pharmacokinetic systems were discovered without any human intervention, which illustrates how a set of distributed machines can contribute scientific knowledge while scaling beyond geographic constraints.

BIOGRAPHICAL SKETCH

Daniel Le Ly was born in Toronto, Ontario on November 14, 1986. Daniel earned his Bachelors of Applied Science with Honours at the University of Toronto in the Engineering Science program. During his undergraduate experience, he helped to design and build two championship racecars with the Formula SAE team, which cultivated his enthusiasm for engineering and design. However, he found his passion for research through undergraduate research opportunities with Professors Charles Lumsden and Paul Chow, made possible by a series of Natural Sciences and Engineering Research Council (NSERC) Undergraduate Student Research Awards.

Daniel completed his Masters of Applied Science in Computer Science at the University of Toronto under the supervision of Professor Paul Chow, with a thesis entitled ‘A High-Performance Reconfigurable Architecture for Restricted Boltzmann Machines’.

Daniel received an NSERC Postgraduate Scholarship and was accepted into the Ph.D. program in the Sibley School of Mechanical and Aerospace Engineering at Cornell University under the supervision of Professor Hod Lipson, and commenced the work presented here.

To my family, for their unconditional love and support.

ACKNOWLEDGEMENTS

This work was supported by a Natural Sciences and Engineering Research Council of Canada (NSERC) Postgraduate Scholarship. This work was also funded by grants from the Defense Threat Reduction Agency (DTRA), U.S. National Institute of Health (NIH), and the National Science Foundation (NSF).

I would also like to extend my deepest gratitude to the following people:

To Professor Hod Lipson for his tutelage, vision, and inspiration throughout the years. Thank you for your guidance in all aspects of my technical and professional development. I am grateful for all the opportunities you have given me.

To the faculty at Cornell University, especially Professors Mark Campbell, Giles Hooker, and Hadas Kress-Gazit for their invaluable discussions, feedback and advice.

To all my collaborators in the Vanderbilt Institute for Integrative Biosystems Research and Education. In particular, I would like to thank Christina Marasco, David McLean, Philip Samson and John Wikswo for their ingenious ideas and commitment to our projects.

To all the members of the Creative Machines Lab who have helped and advised me throughout my work. In particular, I would like to thank John Amend, Jr., Daniel Celluci, Nicholas Cheney, Jeffrey Clune, Paul Grouchy, Apoorva Kiran, Igor Labutov, J. Aaron Lenfestey, Jeffrey Lipton, Robert MacCurdy, Hirotaka Moriguchi, Jonas Neubert, Ethan Ritz, Michael Schmidt, Michael Tolley, and Jason Yosinski for their support and friendship.

To the staff of Cornell University, especially Craig Ryan and Marcia Sawyer.

To my family, for their unconditional love and support. None of this would be possible without you.

TABLE OF CONTENTS

Biographical Sketch	iii
Dedication	iv
Acknowledgements	v
Table of Contents	vi
List of Tables	ix
List of Figures	x
1 Introduction	2
1.1 Overview	2
1.2 Current state of the art	4
1.3 Background	6
1.3.1 Evolutionary computation	7
1.3.2 Symbolic regression	8
1.3.3 Predictor coevolution	10
1.3.4 Multi-objective Pareto optimality	12
2 Coevolutionary search methods and applications	16
2.1 Trainer selection strategies for rank predictors	16
2.1.1 Related work	18
2.1.2 Rank prediction algorithm	20
2.1.3 Experimental setup	28
2.1.4 Experimental results and discussion	33
2.1.5 Conclusion and future work	38
2.2 Optimal experiment design for coevolutionary algorithms using Shan- non information	38
2.2.1 Related work	40
2.2.2 Optimal experiment design via information theory	42
2.2.3 Coevolutionary active learning testbench	53
2.2.4 Testbench results	61
2.2.5 Real-world concrete compression strength experiments	65
2.2.6 Conclusion and future work	68
2.3 Coevolutionary predictors for kinematic pose inference from RGBD im- ages	69
2.3.1 Related work	71
2.3.2 Pose inference algorithm	73
2.3.3 Depth image experiments	82
2.3.4 Conclusion and future work	92
3 Modeling discrete-continuous hybrid dynamical systems	94
3.1 Background	96
3.1.1 Hybrid automata	96
3.1.2 Discrete dynamical system with continuous mappings	99

3.1.3	Related work	100
3.2	Symbolic regression of piecewise functions	102
3.2.1	Problem formalization	102
3.2.2	Clustered symbolic regression	104
3.2.3	Clustered symbolic regression algorithm	109
3.3	Modeling transition conditions	113
3.3.1	Problem definition	113
3.3.2	Related work	114
3.3.3	Transition modeling algorithm	114
3.3.4	Modeling hybrid dynamical systems	116
3.4	Results	121
3.4.1	Experimental details	121
3.4.2	Synthetic data experiments	122
3.4.3	Real data experiment	135
3.4.4	Scalability	138
3.5	Conclusion and future work	139
4	Uncovering hidden dynamical variables	141
4.1	State space transformations of dynamical systems	141
4.1.1	Transformations of nonlinear dynamical systems with one ob- served variable	141
4.1.2	Example realizations for the Lotka-Volterra population dynam- ics system	149
4.1.3	Transformations of nonlinear dynamical systems with multiple observed variables	151
4.1.4	Example realizations for the Chua circuit dynamical system	157
4.2	Inferring ordinary differential equations of arbitrary order	160
4.2.1	Model inference	162
4.2.2	Methods and algorithms	165
4.2.3	Results	175
4.2.4	Conclusion and future work	187
4.3	Discovering simple representations of dynamical systems	188
4.3.1	Related work	189
4.3.2	Inferring differential equation models of arbitrary order	190
4.3.3	Inferring simple state transformations	192
4.3.4	Results	199
4.3.5	Conclusions and future work	214
4.4	Modeling spatial differential systems	215
4.4.1	Symbolic representation	215
4.4.2	Fitness evaluation	217

5	Automated telescience	221
5.1	Related work	223
5.2	Inferring luminescent chemical reactions	225
5.2.1	Background	226
5.2.2	Active learning framework	229
5.2.3	Luciferase reaction model	232
5.2.4	TCPO reaction model	236
5.2.5	Conclusions and future work	242
5.3	Inferring multi-compartment pharmacokinetics models	242
5.3.1	Physical multi-compartment, microfluidic device	244
5.3.2	Experiment design	249
5.3.3	Model inference of discontinuous differential equations	259
5.3.4	Results on a local, synthetic system	264
5.3.5	Remote operated experiments on a physical device	273
5.3.6	Conclusions and future work	286
6	Limitations and future work	287
6.1	Evolutionary computation (Section 1.3.1)	287
6.2	Trainer selection strategies (Section 2.1)	289
6.3	Optimal experiment design (Section 2.2)	289
6.4	Kinematic pose inference from RGBD images (Section 2.3)	291
6.5	Clustered symbolic regression (Section 3.2.2)	292
6.6	Inferring symbolic binary classifiers (Section 3.3)	293
6.7	Inference of hybrid automata (Section 3.2)	294
6.8	Inference of ordinary differential equations of arbitrary order (Section 4.2)	294
6.9	Discovering simple representations of dynamical systems (Section 4.3) .	295
6.10	Inferring luminescent chemical reactions (Section 5.2)	296
6.11	Inferring multi-compartment pharmacokinetics models (Section 5.3) . .	297
	Contributions	299
	Major contributions	299
	Contributions of others	302
	Bibliography	305

LIST OF TABLES

2.1	Glossary of terms for rank predictors	22
2.2	Example rank fitness calculation (Eq. 2.7)	24
2.3	Pose inference algorithm naming convention	84
2.4	Pose inference performance on synthetic images	86
2.5	Pose inference performance on real images	89
3.1	Summary of test data sets for synthetic hybrid systems.	127
4.1	Inference of noisy, synthetic systems	177
4.2	Inference of physical systems	186
4.3	The parameters for generating the synthetic systems	205
5.1	A list of simplified symbols for the luciferase reaction.	230
5.2	The simplified nomenclature for the TCPO reaction according to the compound's role.	239
5.3	The list of commands for device control.	246
5.4	The upper and lower limits of each pump parameter.	250
5.5	Model inference performance on the validation set for the noisy syn- thetic experiments.	272
5.6	The error on the training and validation data sets, as well as the system noise for repeated experiments.	276

LIST OF FIGURES

1.1	Flowchart describing the symbolic regression algorithm.	9
1.2	Symbolic expressions represented as tree structures, with examples of recombination and mutation operations.	10
1.3	An example of the set of solutions provided by symbolic regression. a) The Pareto optimal solutions (in black) are expressions that have the best accuracy for all models of equal or lower complexity. Suboptimal solutions generated by symbolic regression are in grey. b) A plot of the non-dominated solutions with the corresponding the data set. The true model is in bold text, while the remaining solutions are either under- or overfit.	13
2.1	A flowchart of the predictor coevolution process.	21
2.2	The data set, trainers and predictors for the rank prediction example. The data set consists of 101 points generated from a noiseless cubic function. Four trainers, along with their parameter values, are provided. Each predictor consists of only four points from the data set.	23
2.3	Randomly generated expressions of varying complexities.	30
2.4	Normalized mean absolute error of the trainer selection strategies averaged across the test suite versus computational error of the coevolution algorithm. The error bars indicate standard error of the mean ($n = 512$).	34
2.5	Convergence rate of the trainer selection strategies averaged across the test suite versus computational error of the coevolution algorithm. The error bars indicate standard error of the mean ($n = 512$).	35
2.6	The failure percentage and solution error for the strategies at 10^{10} evaluations. The error bars indicate standard error of the mean ($n = 512$).	36
2.7	A flowchart of describing the relationship between the three processes of active learning.	43
2.8	A model inference problem example with five data points and three candidate models. The goal of experiment design is to determine the optimal input, x^* , to disambiguate the candidate models and accelerate the inference process.	44
2.9	The normalized measures of entropy, surprisal and variance for the candidate model example. The optimal input experiment for each criterion is marked with red vertical curves.	52
2.10	A process flowchart for symbolic regression.	55
2.11	A plot of the exponential sinusoid expression.	58
2.12	A plot of the wavelet expression.	58
2.13	A plot of the pulse expression.	59
2.14	The convergence plots for the three testbench expressions and the three noise levels. Error bars indicate standard error ($n = 12$).	62

2.15	The high noise, $SNR = 10\text{dB}$, data set of the median performing run for each design policy overlaid on the ground truth expression. The final normalized error is indicated. Darker points indicate early experiments while lighter points indicate late experiments.	64
2.16	The convergence plot for the three treatments. Note that the full data treatment is a passive learning approach and uses all available data for its training set while the variance and surprisal treatments dynamically add to their training set and the number of instances is equal to the experiment number plus three. Error bars indicate standard error ($n = 32$).	67
2.17	Inferring pose information from a single depth image and an arbitrary kinematic skeleton. The framework is able to pose both (a) quadrupedal spider and (b) humanoid kinematic skeletons without any modifications or training.	70
2.18	a) The acyclic graph representation of the skeleton, b) the intermediate parametric equations and c) the corresponding visual depiction. Link 4 is highlighted for reference.	75
2.19	A visualization of the fitness metric evaluated for a single point. The distance between the point and the nearest surface is computed for each link, indicated by the dashed lines. Of these distances, the shortest length (highlighted) is used for the fitness calculation (Eq. 2.35).	76
2.20	A visualization of the a) mutation and b) recombination operators. Mutation changed the parameters of the highlighted links. For recombination, the root was selected as the crossover point and the link chains were swapped to produce offspring.	79
2.21	An example of predictor co-evolution for pose inference. Two predictors, of four points each, are used to evaluate fitness (Eq. 2.35). Predictor 1 is superior as it obtains the same ranking as the fitness evaluated on the entire data set, while predictor 2 obtains an improper ranking.	80
2.22	Fitness of the best individual vs. computational effort averaged over 128 images. Error bars indicate standard error ($n = 128$).	87
2.23	Pose inference examples on synthetic and real world data. Note the point clouds are not pre-segmented and the colorized links are the result of post-processing for the ease of interpretability.	91
2.24	A histogram indicating the frequency, proportional to color intensity with darker points being selected more frequently, that a point was selected to be used in a predictor.	92

3.1	An example of a hybrid automata model for a simple, 1D driverless car. A schematic of the system is shown in a) and the system diagram represented as a directed graph is shown in b). \mathcal{W} consists of two inputs, u_1 and u_3 , which corresponds to the distance to the nearest sign and vehicle, respectively; while \mathcal{X} consists of the state variables x and \dot{x} , which describe the vehicle's position and velocity. \mathcal{M} consists of three modes, $\{m_1, m_2, m_3\}$, which represent distinct behaviors corresponding to whether the vehicle is approaching a traffic sign, cruising or driving in traffic; and the behaviors for each mode is described by $\{f_1, f_2, f_3\}$. There are five transitions events, each represented by a Boolean condition.	97
3.2	A conversion of the 1D driverless car hybrid automata as a discrete dynamical model with continuous mappings. The system diagram and variable conversion are shown.	100
3.3	An example of the time-series data of membership signals. Transitions are highlighted in grey.	117
3.4	An example of PTP-NTP weight balance. a) Original weight data ($\gamma_{k,n}$). b) Weight data decomposed into $p_{k,n}$ and $n_{k,n}$ signals. c) Scaled $\tilde{n}_{k,n}$ signal. d) $p_{k,n}$ and $\tilde{n}_{k,n}$ recombined to form balanced ($\tilde{\gamma}_{k,n}$).	118
3.5	Schematic diagram of the fully recurrent neural network.	123
3.6	Schematic diagram of the neural network based IOHMM architecture [9]. \mathcal{F}_k are regression networks with a single sigmoidal hidden layer and \mathcal{T}_k are softmax networks with a single sigmoidal hidden layer.	124
3.7	Schematic diagram of an open-loop and closed-loop system in a) and b), respectively.	126
3.8	The system diagram and plots of the noiseless test data sets.	128
3.9	The performance metrics on the four synthetic hybrid systems. Error bars indicate standard error ($n = 10$).	130
3.10	Conversion from program output to hybrid dynamical model for the phototaxic robot with 10% noise. Algebraic simplifications were required to convert program output (a) to inequalities in canonical form (b).	131
3.11	The input-output relationship of the regression networks of NNHMM and symbolic expressions of MMSR (black) overlaid on the Continuous Hysteresis Loop data (grey).	133
3.12	A circuit diagram indicating the two input voltages, v_{GS} and v_{DS} , and the output current i_D , and the measured 3D data plot from the ZVNL4206AV nMOSFET.	136
3.13	The inferred hybrid model compared to the derived expressions.	137
4.1	A flowchart outlining the four different realizations and conditions for transforming between them, using the Lotka-Volterra dynamics as an example (Section 4.1.2).	148

4.2	Process for discerning hidden dynamics. (A) A flowchart illustrating the iterative, symbolic regression process. The cycle (2,3,4) is repeated until a fixed period has elapsed or a predetermined objective error is achieved. (B) A binary tree representation for the third-order, chaotic Duffing oscillator. Implicit differential equations are represented using five operators: input variables, constants, addition, multiplication and differentiation. New candidate expressions are generated by modifying subtrees. (C) Computing an objective error based on the principle of predictability. The algorithm compares the highest order derivative of each expression, which is computed using two separate methods: numerical differentiation and a local root solver.	163
4.3	An example of calculating the fitness of a differential equation model for a damped harmonic oscillator. Beginning with the candidate model (A), the time series data is differentiated numerically (B). The highest order time derivative is calculated using a local root finding method (C) and compared with the numerical derivative for the fitness metric (D). .	173
4.4	Plots of the noiseless data sets for all eleven synthetic system along with their respective differential equations. The state variable x is shown in blue and the state variable y is shown in green.	178
4.5	Plots of the third-order, chaotic Duffing oscillator with various levels of noise corruption.	179
4.6	Accuracy-complexity tradeoff. (A) A plot of the objective error as a function of the computational effort for the third-order, chaotic Duffing oscillator. Error bars indicate standard error ($n = 8$). (B) A plot of the final accuracy-complexity Pareto optimal set for the third-order, chaotic Duffing oscillator. The complexity of true expression for the system is indicated by the dashed line. (C) Selected expressions from the Pareto optimal set in B. The true expression for the system is indicated in boldface.	180
4.7	Plots of the objective error as a function of computational effort for all non-trivial expressions for the synthetic systems. Error bars indicate standard error ($n = 8$).	181
4.8	Plots of the final accuracy-complexity Pareto optimal set for all non-trivial expressions for the synthetic systems. The complexity of true expression for the system is indicated by the dashed line.	183
4.9	The computational time required to infer the third-order chaotic Duffing oscillator. Error bars indicate standard deviation ($n = 8$)	184

4.10	Uncovering hidden dynamical variables. (A) The voltages of the two capacitors were measured over time from a chaotic Chua circuit (B) using a digital oscilloscope. (C) The algorithm automatically searched for differential equation models that described the observed variables and (D) then uncovered transformations of the system that resulted in parsimonious realizations. (E) Without any domain knowledge of circuits or nonlinear dynamics, the algorithm found the standard formulation of the Chua circuit equations and (F) inferred the dynamics of a state variable that corresponded to unmeasured inductor current. Actual circuit, data and results are shown; inductor current is scaled linearly for visualization.	190
4.11	Computational approach for uncovering hidden dynamical variables from experimental data. (A) After inferring a differential equation model using the method described in Section 4.2, search for transformations that result in parsimonious realizations of the differential equation model using a symbolic regression algorithm. (B) Symbolic expressions are represented in computer memory as a tree data structure, where nodes are mathematical operations and leafs represent parameters and variables. The algorithm varies these structures to search the space of transformations. The complexity of an equation is determined by the number of elements in the corresponding tree and the sum of the coefficients weighted by their respective subtrees.	192
4.12	An example of calculating the parsimony fitness for an expression. For the expression and tree structure in (A), (B) illustrates how the fitness is determined: k can be interpreted as the L0-norm of the tree size while p can be interpreted as the L1-norm weighted with the corresponding L0-norm. The binary inequality is shown in (C).	195
4.13	An example of calculating the parsimony fitness for a transformation. The fitness metric and its binary inequality is shown in (A). An example of the fitness for various transforms using the Lotka-Volterra dynamics is shown in (B) while a table of inequality statements is shown in (C). .	197
4.14	An example of resolving an expression using relative precision.	198
4.15	Summary of hidden variables discovered from nonlinear dynamical systems. For each system, the inferred differential equation model and inferred standard state space realization is shown, with the corresponding hidden variable as the red curve.	200

4.16	Alternative representations and computational time. (A) For the Lotka-Volterra system, the space of all transforms is searched and the set of Pareto optimal realizations with respect to parsimony are shown, with the standard state space realization highlighted in red. (B) The time to infer the differential equation model (stage 1) depends primarily on the complexity of the system and the variations in the computational time indicates how likely the algorithm finds approximate solutions. (C) The time to infer the standard realization (stage 2) depends primarily on the complexity of the transformation. Error bars indicate standard deviation ($n = 8$).	202
4.17	The performance plots for the Stage 1 inference of synthetic systems. Error bars indicate standard error ($n = 8$).	206
4.18	The fitness plots as a function of noise. Error bars indicate standard error ($n = 8$).	207
4.19	The search log plot for each synthetic system.	209
4.20	The most parsimonious, nontrivial representations found by the search algorithm.	210
4.21	A plot of the most parsimonious, nontrivial representations found by the search algorithm for each dynamical system. The standard realizations are highlighted in red.	211
4.22	The performance plots for the Stage 1 inference of the Chua circuit. Error bars indicate standard error ($n = 8$).	212
4.23	The search log plot for the transformations in the Chua circuit.	213
4.24	The most parsimonious, nontrivial representations of the Chua circuit found by the search algorithm.	213
4.25	The computational effort and clock time required to infer the differential equation model. Error bars indicate standard deviation ($n = 8$). . .	214
4.26	The computational effort and clock time required to infer the standard realization. Error bars indicate standard deviation ($n = 8$).	214
4.27	The representation of a partial differential equation model in computer memory (A) and its corresponding tree representation (B).	216
5.1	A flowchart illustrating the major components of automated tele-science. The cyclic processes of hypothesis-formation and experimentation are shown. Image courtesy of John Wikswo.	222
5.2	A flowchart of the active learning framework for inferring luciferase bioluminescent reactions.	231
5.3	An example of two experiments specified in a plain-text file. The initial concentrations of each input is defined.	231
5.4	An example of the results of two experiments logged in a plain-text file. The output, v , is a function of the inputs, $\{f, l, a\}$	232
5.5	An example of two inferred models of luciferase enzyme kinetics in a plain-text file.	232
5.6	Experiment parameters for luciferase and ATP.	234

5.7	Experiment parameters for luciferin and ATP.	235
5.8	The luminescent chemistry instrument for inferring the TCPO reaction. The primary components of the device are labeled. Image courtesy of Christina Marasco.	237
5.9	A COMSOL multiphysics simulation of the microfluidic mixing chamber for the TCPO reaction. Image courtesy of Christina Marasco. . . .	238
5.10	The initial setup of the physical, multi-compartment microfluidic device. The microfluidic device filled with dye is shown on the left, while the pump-microscope-computer assembly is shown on the right. Image courtesy of Philip Samson.	244
5.11	The final setup of the physical, multi-compartment microfluidic device. Image courtesy of Philip Samson.	245
5.12	An example of an executable command file. Note that comments are denoted by # and are only embedded in this example for ease of interpretation.	247
5.13	An example of a results file generated from the command file in Fig. 5.12.	248
5.14	An example of the time-series data extracted from a results file.	248
5.15	A diagram of the square wave defined by the corresponding pump parameters.	250
5.16	Experiment design via expected derivatives. The top plot shows the simulated signals generated by numerical integration and the bottom plot shows the derivatives of those signals. Note that the two signals predict different steady state signals, but the derivatives of those signals in the steady state region are both zero. Using expected derivatives, these two models have zero disagreement despite producing entirely different predictions.	254
5.17	Experiment design via weighted mean signal. The top plot shows the simulated signals with the addition of the weighted mean and the bottom plot shows the predicted derivatives based on the mean signal. Note the mean signal predicts a different steady state value than the original signals and as a result, the predicted derivatives using the mean signal are different. This allows the experiment design process to find experiments that makes the models disagree on steady state values.	255
5.18	Erroneous spline fitting for discontinuous inputs. Note the difference between the spline and the true response when the pumps are turned on and off.	261
5.19	Erroneous spline derivatives for trapezoidal waves. The massive discrepancy between the derivatives resulted in equations that did not depend on the pumps.	262
5.20	Spline fitting with with the data segmented according to the input discontinuities.	262
5.21	An example of a data file. This synchronized time-series data with derivatives is obtained by processing the results file in Fig. 5.13 with a smoothing spline.	264

5.22	The 16 repeatability experiments for the noisy synthetic system overlaid on a single plot.	266
5.23	The 5 experiments in the validation set for the noisy synthetic system overlaid on a single plot.	267
5.24	The 19 noisy synthetic experiments in the training set overlaid on a single plot.	268
5.25	The normalized error of the best model measured on the training data for each new noisy synthetic experiments.	269
5.26	The normalized error of the best model measured on the validation data for each new noisy synthetic experiments.	270
5.27	The weighted model variance each new experiment in the noisy synthetic experiments.	271
5.28	The 24 repeatability experiments for the microfluidic device overlaid on a single plot.	274
5.29	The 8 experiments in the training set for the microfluidic device overlaid on a single plot.	275
5.30	The 16 experiments in the validation set for the microfluidic device overlaid on a single plot.	275
5.31	Validation experiment 0.	278
5.32	Validation experiment 1.	279
5.33	Validation experiment 2.	280
5.34	Validation experiment 3.	281
5.35	Validation experiment 4.	282
5.36	Validation experiment 5.	283
5.37	Validation experiment 6.	284
5.38	Validation experiment 7.	285

LIST OF ALGORITHMS

2.1	Non-parametric optimal experiment design algorithm	53
2.2	Evolutionary pose inference algorithm. Details of rank prediction and age-fitness Pareto selection are found in [175, 176].	81
3.1	Generalized expectation-maximization	108
3.2	Clustered symbolic regression	112
3.3	Multi-modal symbolic regression	120
4.1	Integration via finite differences	168
4.2	Integration via black-box, adaptive Runge-Kutta methods	170
4.3	Comparing predicted derivatives	172
4.4	Integration via finite difference stencil	219
5.1	Experiment design for the pharmacokinetic device	259

CHAPTER 1

INTRODUCTION

1.1 Overview

Automated science is an emerging field of research and technology that aims to extend the role of computers in science from a tool that stores and analyzes data to one that generates hypotheses and designs experiments. Despite the tremendous discoveries and advancements brought forth by the scientific method, it is a process that is fundamentally driven by human insight and ingenuity. Automated science aims to develop algorithms, protocols and design philosophies that are capable of automating the scientific process. Instead of relying on first principles to generate hypotheses, a data driven approach is taken, which attempts to reverse engineer the same hypothesis from observations and infer underlying casual mechanisms.

By incorporating a more active role in the scientific process, automated computer and robotic systems can achieve a revolutionary role in scientific discovery. As the systems of interest increase in complexity and scope, building scientific models of these systems is a challenging task for both individual as well as groups of scientists. Automated science aims to accelerate this discovery process by designing experiments and extracting information in an optimal fashion, working ceaselessly without fatigue and achieving high-throughput analysis through parallelization.

This dissertation presents advances and contributions to the field of automated science and addresses three core areas. The first core area focuses on advances and applications of coevolutionary algorithms, which provides the computational engine in the other core areas, and is described in Chapter 2. Section 2.1 presents trainer selection strategies

for rank predictors and is based on a paper published at the 2011 IEEE Congress of Evolutionary Computation [119]. Section 2.2 discusses an approach for optimal experiment design using coevolutionary algorithms based on Shannon information and is based on a paper published in IEEE Transactions on Evolutionary Computation [122]. Section 2.3 presents an application of coevolutionary predictors for kinematic pose inference from RGBD images and is based on a paper presented at the 2011 RGB-D Workshop [123] and published at the 2012 Genetic and Evolutionary Computation Conference [120].

The second core area focuses on inferring the underlying structure of dynamical systems and is divided into two chapters. Chapter 3 describes an approach to infer discrete-continuous hybrid dynamical systems and is based on a paper published in Journal of Machine Learning Research [121]. Chapter 4 presents an approach to uncover hidden state variables of dynamical systems from time series data. Section 4.2 describes an approach to inferring ordinary differential equations of arbitrary order and is based on a paper under review in the Proceedings of National Academy of Sciences while Section 4.3 discusses how hidden state variables can be uncovered from differential equations and is based on a paper under review in Science.

The final core area focuses on experimental work in the field of automated tele-science. Chapter 5 describes a collection of applications of remotely controlled devices designed to discover scientific models of dynamical systems.

Chapter 6 summarizes the limitations of each algorithm presented in this dissertation and outlines avenues for future work.

1.2 Current state of the art

The idea of automating the scientific process is a relatively new concept brought forth by the advancements of computer science and the increasing availability of computational power [105, 107]. As a result, the field itself is referred to by a series of different names, including ‘automated science’ [211], ‘machine science’ [52], and ‘robot scientist’ [98], all of which may be used interchangeably.

One of the first reports of automated science was DENDRAL [115], which attempted “to embody the strategy of using detailed, task-specific knowledge about the problem domain as a source of heuristics, and to seek generality through automating the acquisition of such knowledge”. It was an expert system that contained knowledge of chemistry and modified that knowledge base to analyze experimental mass spectra data. The goal of the project was to determine the structure of organic chemical compounds and was one of the first applications of automated scientific reasoning. DENDRAL consisted of a collection of programs, organized into “planning”, “generating”, and “testing” operations. Despite the potential of the project, the impact of the work was limited for a variety of reasons, including it underperformed compared to human counterparts and was relegated primarily to a role of “solution-checker”. This set of programs was also extended into the software suite Meta-DENDRAL [26, 55], which accepted additional inputs regarding the mass spectra and structure pairs.

Other early examples of automated science include the Automated Mathematician [110], which attempted to discover math concepts by representing characteristic functions in a functional programming paradigm. This approach was later extended into more general, heuristic based learning program named EURISKO [110], which was able to optimize the design of microchip integrated circuits. KEKADA was another

early, heuristic based system that was able to discover the urea synthesis pathway [105]. A series of early approaches aimed to discovery scientific laws as algebraic equations, which include ABACUS [53], BACON [108], Fahrenheit [217] and IDS [139]. A survey of this topic is reviewed by Sparkes et al. [189].

These early systems were limited by two primary constraints. First, the scope of models was relatively limited and often required domain-specific heuristics and assumptions. However, more recent work has expanded the capabilities of these approaches. Schmidt and Lipson presented a data driven system that is able to extract natural laws of momentum and energy from motion-tracked observations [173]. Without any prior knowledge of physics or geometry, the algorithm successfully inferred meaningful models of physical systems, including the Hamiltonian Law for the chaotic double pendulum.

Second, the early approaches often processed a static data set and did not design or execute their own experiments through active interactions with their environments. Recent work in this field has resulted in significant advances in the field of automated science. Bongard et al. showed that a robot could learn a resilient self-model through continuous experimentation [18]. By following an iterative process of modeling kinematic structures from data and designing motor commands that best disambiguates the competing models, the robot was able to create a sufficiently accurate model, which was later used for gait discovery. Following a similar process, Bongard and Lipson designed an iterative approach to discovering the structure of nonlinear dynamical systems by designing the initial conditions for a variety of systems [17]. Finally, King et al. was able to identify genes encoding ‘locally orphan enzymes’ in the yeast *Saccharomyces cerevisiae* through a robotic system, named ‘Adam’ [98]. The automated laboratory

robotic system used formal logical expressions to analyze a bioinformatic database and homology search techniques to hypothesize likely candidate genes for discovery.

Due to the recent explosion of high impact developments in the field of automated science, the reactions have been cautiously optimistic while also outlining areas for future growth. Many opinions highlight the potential of automated science to deal with the exponential increase of available data with respect to both the volume [31, 196], as well as the variety of data formats and structures [133]. Some perspectives discuss the challenges of scalability and need for advanced methods [134], while others attempt to highlight the near-term and long-term roles for automated science [52, 211].

However, not all opinions of automated science are as positive. Gianfelici argues that “‘Machine science’ could more accurately be considered ‘machine-aided science’ [...] until a machine can produce results can produce results of solve open problems without human direction” [66]. Similarly, there several concerns regarding the need for human interpretation and the irreplaceable role of humans in science [76, 111]. Finally, Anderson and Abrahams argues that computers are unable to replicate the “creative, exploratory nature of true science” [5].

1.3 Background

This section introduces essential information regarding background concepts and previous research that are referenced in several chapters of the text. Extracting models for dynamic systems from experimental data is an NP hard computational problem [40], and as a result, much of the work presented in this dissertation builds upon heuristic approaches. This section will cover evolutionary computation, symbolic regression, predictor coevolution, and multiobjective Pareto optimization.

1.3.1 Evolutionary computation

Evolutionary computation is a heuristic approach that was originally inspired by biological evolution and Darwinian selection [58, 60]. It is a population-based algorithm which contains a collection of individuals, or candidate solutions, that compete to survive in a simulated environment. The algorithm begins with initially random individuals and their fitness is evaluated according to a predefined objective function. The algorithm proceeds in an iterative fashion where new individuals are generated using stochastic, biologically-inspired operations and a new generation of individuals are selected according to their fitness evaluations. The program then terminates after a predefined condition is met, which usually occurs once a prespecified computational effort or a desired fitness is achieved.

Evolutionary algorithms consists of four, major components:

1. **Representation** – the structure and parameterization of an individual or candidate solution. From a biological evolutionary perspective, the representation corresponds to the genotype or genetic makeup of the individual.
2. **Evolutionary operators** – the methods and processes by which new individuals are produced. The traditional operators are recombination (also known as crossover) and mutation. Recombination consists of generating new individuals by inheriting genetic material of stochastically varying amounts from a set of parents, while mutation consists of stochastically modifying portions of the genetic makeup to produce a new individual.
3. **Fitness evaluation** – the numeric evaluation of an individual. From a biological evolutionary perspective, the fitness evaluation corresponds to the phenotypic

performance of the individual. For evolutionary algorithms, fitnesses are typically predefined using an objective error function.

4. **Selection** – the mechanism by which individuals are removed from the population. From a biological evolutionary perspective, selection removes relatively poorer individuals from the genetic pool, allowing for superior traits to persist in the population. Selection usually involves comparing the fitnesses of individuals, although the method of comparison is nontrivial and can have a significant effect on the performance of the algorithm.

Evolutionary computation is a broad field, which includes popular techniques such as genetic algorithms [78, 68], genetic programming [104, 186] and particle swarm optimization [96].

1.3.2 Symbolic regression

Symbolic regression is an established search method to find analytical mathematical descriptions of experimental data based on genetic programming [104]. Unlike traditional regression approaches that begin with equations of a prespecified form and fit unknown model parameters, symbolic regression searches for both the parameters and equation structure simultaneously. The algorithm begins by randomly combining mathematical building blocks, such as algebraic operators, constants and observed variables, to form an initial set of expressions. New candidate expressions are generated by probabilistically recombining components of previous expressions and through random variations of subexpressions. The algorithm then evaluates how well each expression models the experimental data, and retains superior models for subsequent iterations while abandoning unpromising candidates. The algorithm terminates after a desired level of accuracy

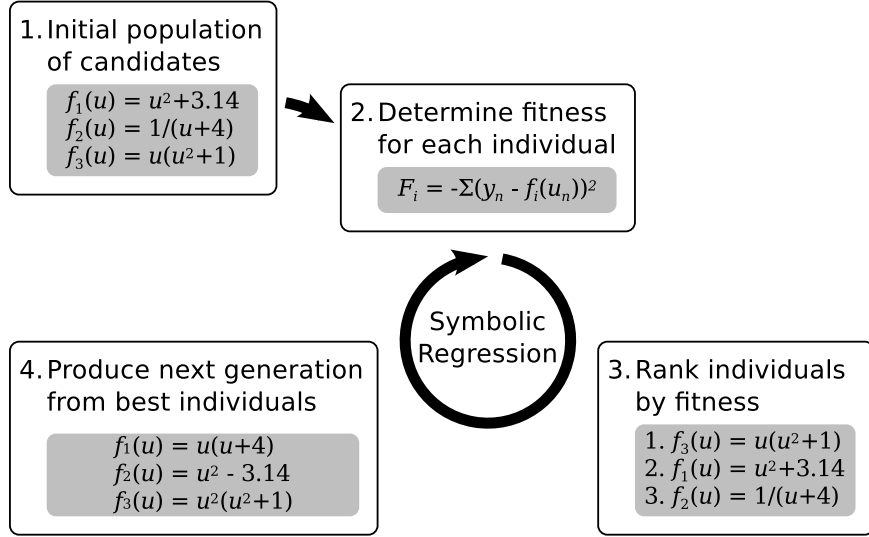


Figure 1.1: Flowchart describing the symbolic regression algorithm.

is achieved and returns the set of expressions that are most likely to correspond to the underlying mechanisms of the system. This process is summarized in Fig. 1.1.

Symbolic expressions are represented as free-form lists of operations and parameters, as thus, both the form of the equation and its parameters are part of the search process. There are a variety of representations of expressions, including binary trees that consist of algebraic operations with numerical constants and symbolic variables at its leafs [42, 128], as well as acyclic graphs [171] and tree-adjunct grammars [138]. Each representation has its own tradeoffs – unless otherwise stated, the binary tree representation was chosen for the work presented in this dissertation due its ease of implementation and manipulation.

The tree structure provides a terse representation of symbolic functions that is capable of representing a wide range of plausible expressions – for example, complex expressions, including non-linear and rational equations, can be easily represented. Generating random trees for initialization generally involves randomly filling the tree with nodes, sampling random variables and constants where necessary, allowing for arbitrar-

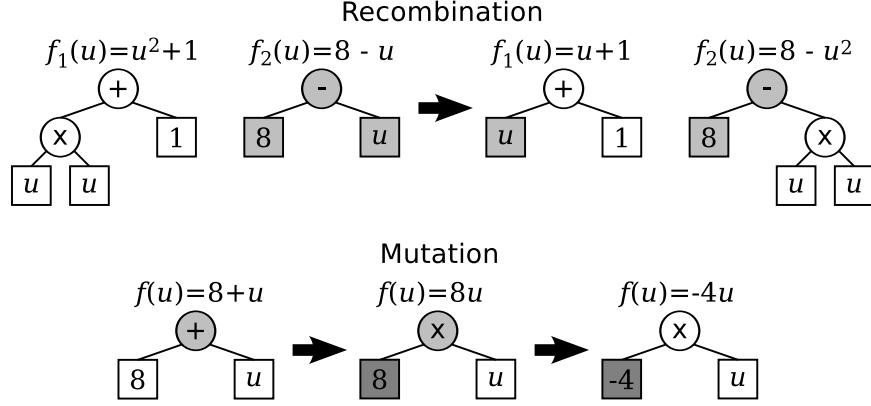


Figure 1.2: Symbolic expressions represented as tree structures, with examples of recombination and mutation operations.

ily sized expressions. Evaluating an expression requires simply substituting values from the data set, traversing the tree and computing well-defined operations. Furthermore, the tree structure is extremely amenable to the evolutionary processes of recombination and mutation through tree manipulations (Fig. 1.2).

1.3.3 Predictor coevolution

A common criticism and prohibitive limitation in evolutionary computation stems from the computational demands of these algorithms, which generally arises from fitness calculations [146]. Often determining the fitness of a single individual requires repeated evaluation of a local metric for each datum in the data set [188]. For example, defining the fitness as the mean squared error requires computing the squared error for each datum individually. Although this scales linearly with the size of the data set, it nonetheless presents a critical limitation when the data set, population size, or number of generations becomes large.

An efficient approach to alleviate the computational requirements involves approximating fitnesses [89]. Rather than calculating the true fitnesses across the entire data set, a coarser and lightweight approximation is substituted. A particularly effective method to improve performance is by coevolving predictors [172, 176]. Instead of using the complete data set, the fitness is measured only on a dynamic subset of the data. The members of this subset, called predictors, are coevolved simultaneously based on the solution population, allowing for evolutionary progress through direct competition. Surprisingly, large data sets can be sufficiently modeled by a subset that are orders of magnitude smaller, resulting in drastic improvements in performance. In a symbolic regression problem, data sets that consist of thousands of points and tens of variables can be effectively replaced by a dynamic selection of just four points [172].

Schmidt and Lipson first presented the idea of fitness predictors [172]. These predictors referenced a subset of the data and were evolved to directly approximate the fitness of a collection candidate solutions. The candidate solutions, in turn, were evolved to model the predictor subset, resulting into two populations in competitive evolution with antagonistic goals. Schmidt and Lipson later revised their approach using rank predictors [176]. Rather than rewarding predictors for their ability to predict fitnesses directly, they were rewarded only for their ability to produce the same ranking of the candidate solutions. Since selection processes often compare individuals by ranking, rewarding predictors based on their ability to rank individuals, as opposed to predict their fitness directly, proved to generate significantly more robust results and faster convergence times.

1.3.4 Multi-objective Pareto optimality

In evolutionary computation, it is often important to consider more than a single fitness criteria. Some approaches combine the multiple objectives into a single fitness, often with a weighted sum function [11]. However, the weight balance is often not known in advance, and as a result, this approach adds another set of parameters and complexity to the search problem.

Instead, a multi-objective approach that values each criteria independently provides a powerful tool in evolutionary computation. In particular, approaches based on Pareto optimality are useful [106]. Pareto optimization is based on the notion of partially ordered sets with each objective as an independent axis, as described in Definition 1.3.1.

Definition 1.3.1. Pareto domination: *For a candidate solution x , let $\mathbf{y} \in \{y_1, \dots, y_N\}$ be an image of x , where y_n corresponds to a single fitness criterion and y_n belongs to a fully ordered set. The solution x^* **dominates** the solution x if and only if $y_n^* \geq y_n$ for all n . Furthermore, the solution x^* **strictly dominates** the solution x if and only if $y_n^* > y_n$ for all n .*

Multi-objective Pareto optimality is used in this work as both post-analysis tool for model overfitting (fitness-complexity Pareto analysis), as well as the central principle for a selection approach (age-fitness Pareto selection).

Fitness-complexity Pareto analysis

An important quality of symbolic regression is its natural method to deal with overfitting, where the inferred model captures the peculiarities of the data set rather than the underlying truth. Overfitting is a major issue in evolutionary computation and ma-

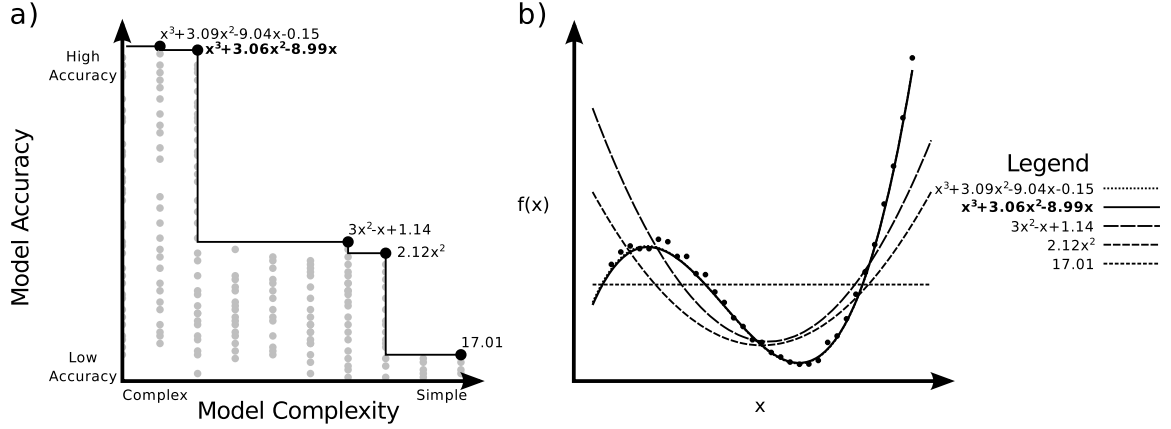


Figure 1.3: An example of the set of solutions provided by symbolic regression. a) The Pareto optimal solutions (in black) are expressions that have the best accuracy for all models of equal or lower complexity. Suboptimal solutions generated by symbolic regression are in grey. b) A plot of the non-dominated solutions with the corresponding the data set. The true model is in bold text, while the remaining solutions are either under- or overfit.

chine learning [11]. In symbolic expressions, overfitting occurs by inferring a model with greater complexity than the ground truth – for example, a cubic model is used to fit a quadratic function. Thus, there is a fundamental trade-off between the accuracy and complexity of a candidate model, where overfitting incurs additional complexity to accurately model the noise contributions in the data.

The inherent population dynamics of evolutionary computation is leveraged to provide a multi-objective approach to dealing with overfitting. By design, symbolic regression generates numerous candidate models with varying degrees of complexity and accuracy. Rather than considering every generated expression as a candidate model, individual expressions are compared against a continuously updated, multi-objective record. This approach, called Pareto optimization, forms a set of non-dominated solutions which provide the best fitness for a given complexity (Fig. 1.3). This method reformulates the problem of overfitting as model selection along the accuracy and com-

plexity trade-off. As Pareto optimization is a post-processing technique that analyzes expressions only after they have been generated by symbolic regression, it does not interfere with the underlying search process.

Age-fitness Pareto selection

The concept of multi-objective Pareto optimization can also be leveraged for a selection operator based on age dynamics. A primary concern in evolutionary computation is premature convergence, which occurs when the population collapses on to a local optimum and no longer explores alternative solutions. This failure mode can be remedied by initiating random restarts, which involves removing the entire population and reseeding with random individuals in hope that a new optimum will be found. Unfortunately, random restart approaches are extremely inefficient as they waste all of the previous computational progress. Furthermore, it is difficult to know when to initiate a random restart since evolutionary algorithms are heuristic and their performance is not guaranteed.

A naive solution to premature convergence is to add a randomly generated individual to the population every generation. Although this may appear to simulate a random restart, these new random individuals are extremely likely to be selected against, since they are competing with individuals who had evolutionary time to achieve superior fitnesses. This is a particularly troublesome issue for new individuals who would have reached the global optimum, but were prematurely removed from the population because they were competing against superior individuals who already reached a local optimum.

Hornby first presented the idea of using Age Layered Population Structures (ALPS) as a method of random restarts while protecting the new individuals [79, 80]. This ap-

proach bracketed the population into smaller groups, divided by a notion called genetic age. Every individual only competed against other individuals within the same bracket, or age layer. After a predetermined number of generations, the best set of individuals were promoted to the next layer and the lowest layer accepted a number of randomly generated individuals.

This approach is analogous to the development of athletes: young athletes are bracketed by age and only compete against other athletes in the same division. As they improve in skill, they are promoted to higher tiers of competition. The tiered level allows young talent to develop since if all the athletes were to compete in a single pool, new candidates would never be able to compete with the well-established athletes.

However, the age layered approach requires a large number of parameters to be hand-tuned, including the number of layers, the population size of each layer, the number of promoted solutions and the rate at which promotion occurs. Schmidt and Lipson built upon the idea of genetic age, but instead presented the approach as a continuous Pareto optimization [175, 176]. In this approach, genetic age is a count that measures how long the genetic material in the solution has existed and a random, age-zero individual is inserted into the population every generation. Selection occurs by randomly removing solutions that are non-dominated according to the age-fitness Pareto dimensions. Thus, solutions are removed if and only if there are younger solutions that have a superior fitness. This Pareto approach extends the layered approach by removing need for hand-tuned parameters, while maintaining the age protection required for continuous random restarts to prevent premature convergence.

CHAPTER 2

COEVOLUTIONARY SEARCH METHODS AND APPLICATIONS

This chapter focuses on a variety of search methods and applications for coevolutionary algorithms. The first section presents trainer selection strategies for rank predictor coevolution while the second section discusses optimal experiment design for coevolutionary algorithms. The third section presents an application of coevolutionary algorithms for kinematic pose inference of RGBD images.

2.1 Trainer selection strategies for rank predictors

Evolutionary algorithms are popular, stochastic, population-based, heuristic algorithms that have been successfully applied to a wide range of optimization and search problems [153]. Despite the variety of applications, a common criticism and prohibitive limitation in practice stems from the computationally heavy demands of these algorithms [89]. Often, the primary culprit in the computational requirements arises from calculating fitnesses [146]. Evolutionary algorithms operate by incrementally improving a population via iteratively selecting the most fit individuals. In many applications, determining the fitness of a single individual requires repeatedly evaluating a local metric over large data sets [188].

A general approach to alleviate the computational requirements involves modeling and approximating fitnesses [89]. Rather than calculating expensive fitnesses across large data sets, a coarser and lightweight approximation can often be substituted. A particularly effective method to improve performance is by coevolving predictors [172, 176]. Instead of using the complete data set, the fitness is measured only on a dynamic subset of the data. The members of this subset, called *predictors*, are

coevolved simultaneously based on the solution population, allowing for evolutionary progress through direct competition. Surprisingly, large data sets can be sufficiently modeled by a subset that are orders of magnitude smaller, resulting in drastic improvements in performance. In a symbolic regression problem, data sets that consist of thousands of points and tens of variables can be effectively replaced by a dynamic selection of just four points [172].

The key to this coevolution technique relies on the systematic method of evolving predictors. First, predictors were rewarded on their ability to predict the exact fitness [172]. However, it was shown that rewarding predictors based on their ability to rank solutions, rather than using fitness measurements directly, proved to be a far superior method [176]. Although this extremely coarse approximation almost certainly provides inaccurate fitnesses, it is still capable of promoting evolutionary progress and, in fact, results in a 5-fold performance improvement over fitness predictors.

However, an often overlooked but equally essential component of these predictor coevolution algorithms is the *trainer* selection strategy. The trainer population is a “hall of fame” of solution individuals which are used to evolve the predictors, and thus are integral in the emergent behavior of the system. Despite their critical role, strategies for selecting trainers have not been thoroughly investigated and compared.

This section presents four different trainer selection strategies for coevolving rank predictors. The performance of these strategies are tested and compared on hundreds of randomly generated test problems over a range of difficulties.

The rest of this section is organized as follows: Section 2.1.1 discusses the related work in fitness approximation and coevolution. Section 2.1.2 describes the rank prediction algorithm and the four trainer selection strategies. The experimental setup is

detailed in Section 2.1.3 and the results are presented and analyzed in Section 2.1.4. This work is summarized in Section 2.1.5.

2.1.1 Related work

This work draws from two distinct fields: fitness modeling and competitive coevolutionary algorithms. Fitness modeling is a well understood technique of using predefined models or coarse simulations to approximate fitness calculations in evolutionary algorithms with the primary goal of reducing the computational overhead [89]. Although they are most commonly used to increase the performance of fitness evaluations, fitness approximations have also found alternative uses in other fields such as approximating fitnesses where no explicit model exists [92], dealing with noisy fitness functions [22] and modeling multi-modal fitness landscapes [113].

There are several different techniques for approximating fitness values as a part of the evolutionary algorithm. The two most prominent methods are fitness inheritance and fitness imitation. In fitness inheritance, fitness computations are saved by estimating the fitness of offspring individuals based on their parents during the recombination process [185]. This approach builds a model of the fitness function based on the probabilistic components of the algorithm [27, 151]. The second class is fitness imitation, which is built on the assumption that similar genotypes, or solution representations, will share similar phenotypes, or solution fitnesses. The first step is to cluster individuals into groups based on genotypic distance. Only the fitness of a representative individual of each cluster is computed directly and it is assigned to the other individuals in that cluster [91, 97].

In contrast, fitness predictors are a class of fitness modeling approaches that are so coarse that they are unable to approximate the entire fitness landscape. Instead, predictors are coevolved with the solution population to cater to the evolutionary process via direct competition. Predictors reduce the computational overhead of fitness evaluations by subsampling the training data [3, 147, 172, 199]. Since the fitness metric of many problems is often defined as a sum of local errors, this subsampling approach ignores the majority of the data with the exception of points described by the systematically evolved predictors.

The second area of relevant research consists of coevolutionary algorithms – heuristic approaches which use multiple populations with interdependent fitnesses. In competitive coevolution, the populations reciprocally drive each other to increasing levels of performance and complexity [165]. However, for non-trivial problems, a persisting challenge arises from the computational complexity of mixed fitness assessment between large populations. A popular solution to this issue is the “hall of fame” technique, where a subset of good individuals from prior generations are saved and used for comparison [112, 165].

In this work, rank predictors are used in a competitive coevolutionary algorithm to subsample the training data and are rewarded for their ability to accurately rank solutions [176]. Previous work in this field focused on developing heuristics for evolving predictors, while this work investigates trainer selection strategies. Trainers act as a “hall of fame” for the solution population and the strategies are selection methods to populate the hall.

2.1.2 Rank prediction algorithm

This section begins with an overview of the predictor coevolution framework followed by an illustrative example of rank predictors in a regression problem. Four trainer selection strategies are then presented and discussed.

Predictor coevolution framework

A traditional evolutionary algorithm can be viewed as an optimization problem to find the most fit *solution* to a given data set. Formally, given a space of candidate solutions, \mathbb{S} , the optimal solution, s^* is defined as:

$$s^* = \operatorname{argmax}_{s \in \mathbb{S}} F(s; D) \quad (2.1)$$

where $F(s; D)$ is the *solution fitness* of the candidate solution s evaluated on the training data set $D = \{d_1, \dots, d_N\}$. When the solution fitness is evaluated on the complete data set, the result is called the *exact fitness*. Evolutionary techniques maintain a population of candidate solutions, \mathcal{S} , and the algorithm proceeds by selecting the most fit individuals and recombines them iteratively with the goal of producing increasingly fit solutions in each generation.

In the predictor coevolution algorithms, two additional populations are introduced – *predictors* and *trainers* – which results in several key changes to the underlying optimization problem. First, solutions are no longer evolved to optimize exact fitnesses. Rather, the solution fitness is evaluated only on the current best predictor; the resulting fitness is called the *predictor fitness*. Since a predictor has drastically fewer points than the complete data set, it is significantly faster to calculate predictor fitnesses than exact fitnesses, resulting in large performance gains.

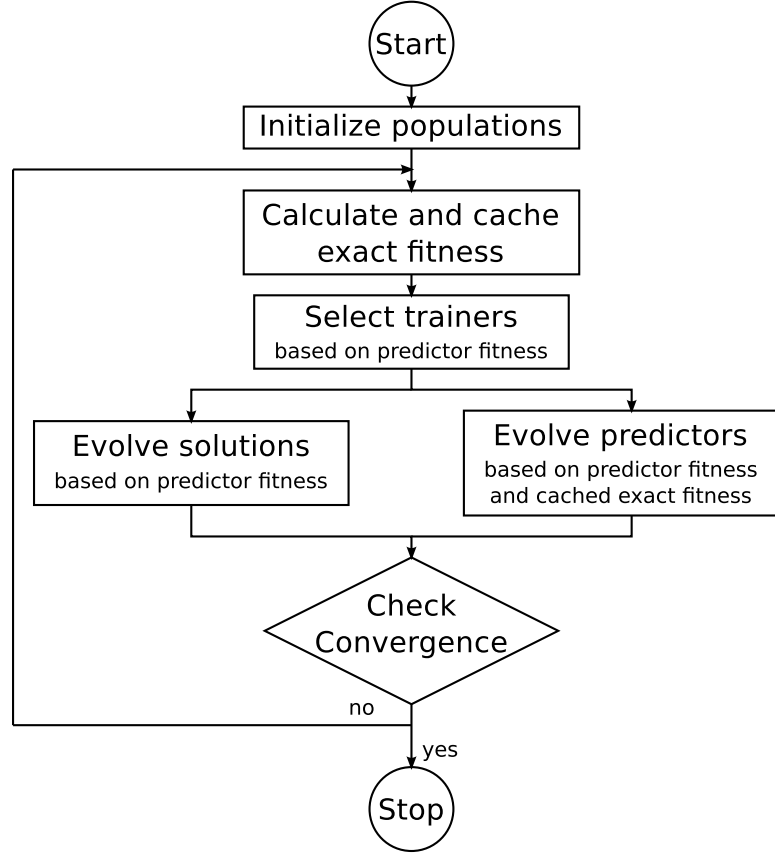


Figure 2.1: A flowchart of the predictor coevolution process.

Next, the trainer population, \mathcal{T} , is a “hall of fame” of solutions selected according to a *trainer strategy*, $T(\cdot)$, to aid in the evolution of predictors. Thus, the trainer strategy’s role is to select solutions from the population based on the current predictor population.

The final population is the predictor population, \mathcal{P} . Each predictor is a small subset of the complete data set, $p = \{d_1, \dots, d_K\} \subset D$. Predictors are coevolved with the solutions based on the *predictor fitness*, $R(\cdot)$, which fundamentally is a comparison between predictor fitnesses and exact fitnesses evaluated using the trainer population. Thus, the evolutionary process rewards predictors that can accurately predict the behavior of the

Exact fitness	The solution fitness measured on the complete data set
Predictor population	The secondary population used for fitness approximation and competitive coevolution
Predictor fitness	An objective function that prescribes the optimality of the predictor population
Rank predictors	Predictors whose fitness is a function of their ability to rank solutions
Solution population	The primary optimization population
Solution fitness	An objective function that prescribes the optimality of the solution population
Trainer population	A tertiary population of selected solutions used to train predictors
Trainer strategy	An algorithm to select trainers from the solution population

Table 2.1: Glossary of terms for rank predictors

trainer population. The three evolutionary rules are formally defined as follows:

$$s^* = \operatorname{argmax}_{s \in \mathbb{S}} F(s; p^*) \quad (2.2)$$

$$t^* = \operatorname{argmax}_{s \in \mathcal{S}} T(F(s; p)) \quad , \quad \forall p \in \mathcal{P} \quad (2.3)$$

$$p^* = \operatorname{argmax}_{p \in \mathcal{D}} R(F(t; p), F(t; D)) \quad , \quad \forall t \in \mathcal{T} \quad (2.4)$$

The predictor coevolution process is summarized in Fig. 2.1 and a glossary of terms is included in Table 2.1. This framework provides several benefits compared to a traditional implementation of evolutionary algorithms. The primary benefit is that the expensive exact fitness calculations are replaced with the light-weight predictor fitness calculations. In predictor coevolution, exact fitnesses are only computed when the trainer population is updated; these values are calculated once and cached for the predictor evolution until the next update. Other benefits include faster convergence rates and reduced bloat [172].

Rank prediction example

This subsection illustrates the theoretical predictor framework via a regression example and depicts how rank predictors are implemented. The optimization problem is: given the data set shown in Fig. 2.2, find the parameters of a cubic function that minimizes the normalized mean absolute error. For a data point $d_n = (x_n, y_n)$, the following relationship is assumed:

$$y = \sum_{i=0}^4 w_i x^i \quad (2.5)$$

As a result, each solution represents a collection of weight parameters, $s = \{w_0, \dots, w_4\}$, and the optimization problem is to find the parameter values that produce the highest fitness.

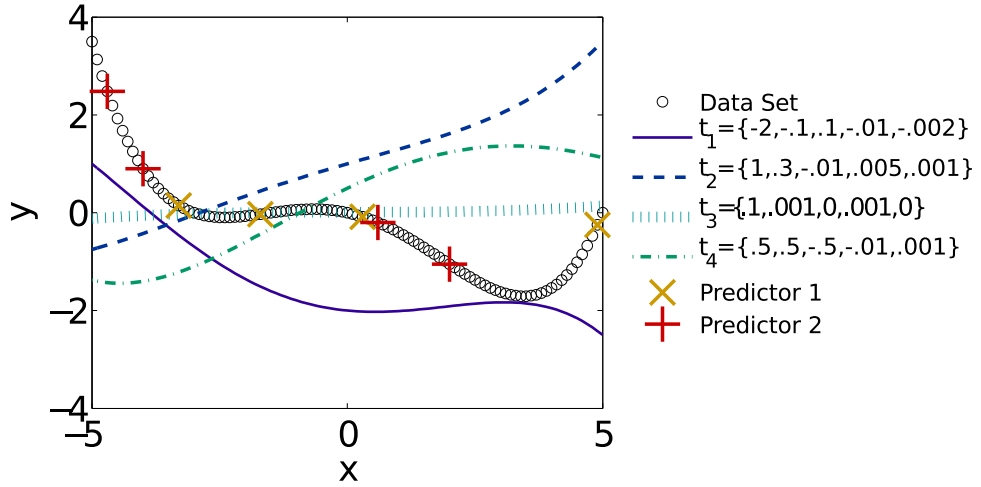


Figure 2.2: The data set, trainers and predictors for the rank prediction example. The data set consists of 101 points generated from a noiseless cubic function. Four trainers, along with their parameter values, are provided. Each predictor consists of only four points from the data set.

Let $\hat{y}_n(s, x_n)$ be the prediction from a candidate solution s on data point n , then the solution fitness is calculated as the normalized mean absolute error:

$$F(s; p) = -\frac{1}{K} \sum_{k=1}^K \frac{|\hat{y}_k - y_k|}{\sigma_y} \quad (2.6)$$

Trainer		Predictor 1			Predictor 2		
Rank	Exact Fitness	Rank	Predictor Fitness	Rank Error	Rank	Predictor Fitness	Rank Error
t_3	-0.78	t_3	-0.17	0	t_3	-1.20	0
t_1	-1.18	t_4	-0.97	1	t_1	-1.29	0
t_4	-1.71	t_2	-1.40	1	t_2	-2.13	1
t_2	-1.95	t_1	-1.55	2	t_4	-2.37	1
		Rank Fitness:		-4	Rank Fitness:		-2

Table 2.2: Example rank fitness calculation (Eq. 2.7)

where σ_y is the standard deviation of y . Note that the solutions are evolved using only the predictor fitness.

Trainers are a dynamic collection of solutions used to evolve predictors. In this example, an arbitrary set of trainers are provided (Fig. 2.2). However, strategies to select trainers from the solution population are described in Section 2.1.2.

Predictors are a subset of the data. Two examples of predictors are shown in Fig. 2.2. In rank prediction, the exact and predictor fitnesses are calculated and used to rank the trainers. The rank fitness is then calculated as the Manhattan distance between the ranks of each trainer:

$$R(F(t; p), F(t; D)) = - \sum_{i=1}^{|T|} \left| \text{rank}[F(t_i; p)] - \text{rank}[F(t_i; D)] \right| \quad (2.7)$$

This rank fitness metric rewards predictors that can correctly predict the same ranking of trainers as produced by the exact fitnesses. This metric is similar to the previous metric described in [176] but provides a smoother fitness landscape by using a distance-based metric rather than a Boolean-based metric. An example of rank fitness calculations is shown in Table 2.2. Note that the predictor fitnesses are not used directly and only the ranking is used to determine rank fitnesses. In this example, all the values of predictor 1 lie near the zeros of the cubic function and thus it has limited ranking capabilities. In contrast, predictor 2 highlights some of the more relevant features of the data

set and is superior to predictor 1 as it obtains a higher rank fitness for the given trainer population.

Trainer selection

Trainers are a population of solutions which are used to evolve rank predictors, and consequently, play a critical role in the emergent behavior of the coevolved system. A diverse and meaningful trainer population encourages effective predictors while a monotonic trainer population provides no gradient for predictor evolution and greatly stifles the evolutionary progress. However, the optimal strategy to select trainers from the solution population is not obvious and requires further investigation.

A series of heuristic, rule-based strategies are presented for trainer selection. A primary criteria in these trainer selection strategies is a low computational overhead since the ultimate goal is to increase the performance of evolutionary algorithms. Four different strategies, representing a range of concepts and complexities, are investigated:

1. **Random selection** [*RAN*] – This strategy is the simplest and most naive method to populate the trainer population, providing a baseline for comparing strategies. Given a solution population, select solutions randomly with a uniform distribution:

$$T_{\text{RAN}}(F(s; p)) = U(1, |S|) \quad (2.8)$$

2. **Rank variance** [*VAR*] – A more methodological approach to selecting solutions is to pick solutions with the highest variance in rank. Given a predictor population, rank all of the solutions for each predictor and pick the solutions that have the

greatest discrepancies in the rank:

$$T_{\text{VAR}}(F(s; p)) = - \sum_{i=1}^{|\mathcal{P}|} (\text{rank}[F(s; p_i)] - \langle \text{rank}[F(s; p_i)] \rangle)^2 \quad (2.9)$$

where $\langle \cdot \rangle$ denotes the mean value. This strategy is closely related to previous work on using coevolution to guide the search by exploring areas of least confidence [16, 90]. Variance provides an inverse measure of rank prediction confidence, and thus, selecting solutions with the highest variance may provide a better guide to the predictor evolution.

3. **Rank variance with memory [MEM]** – This strategy builds upon the *VAR* approach, except the trainer population is treated with memory. Rather than completely replacing the entire trainer population in a single update, only a single trainer is switched. The trainers are kept in a queue data structure. In each update, the oldest trainer is removed and is replaced by a new trainer of the highest variance in the current population. The motivation of this approach is to minimize drastic changes to the trainer population, since this may adversely affect the predictor evolution.
4. **Rank-variance/exact-fitness Pareto optimal [RFP]** – The final trainer selection strategy is the most complex approach, which again leverages the concepts from the *VAR* strategy. Although fitness rankings may vary widely based on the predictor, the rank of each solution is not completely independent and there are noticeable correlations and similarities amongst the rankings. By the nature of variance measurements, the solutions with the highest variance are likely those around the median rank. Thus, strategies that rely solely on variance are often comprised of trainers of median rank.

While this distribution of trainers is not an issue by itself, it must be analyzed within the context of an important observation: predictor coevolution does not

guarantee elitism on exact fitnesses even with elitist selection methods for solution evolution. Elitism is the property that the most fit solution must always exist in the next generation. However, since solutions are evolved only using the predictors, elitism can only be guaranteed with respect to predictor fitnesses, not exact fitnesses.

This strategy attempts to imitate elitism by ensuring that solutions with high exact fitnesses are not underrepresented in the trainer population. This attempts to balance the information content of the population, or solutions of high variance, and the significance of the population, or solutions of high exact fitness.

Similar to the previous approaches, given a solution population, the rank variance is computed. However, trainers are prioritized based on their exact fitness and whether they are rank-variance/exact-fitness Pareto optimal – that is, solutions that provide the highest variance for a given exact fitness. However, since the number of Pareto optimal solutions may not equal the size of the trainer population, trainers are selected according to the following priorities:

- i. The solution with the highest exact fitness
- ii. Solutions that are rank-variance/exact-fitness Pareto optimal, beginning with the highest variance
- iii. Solutions that are not rank-variance/exact-fitness Pareto optimal, beginning with the highest variance

Note that, despite the design considerations, this strategy does not guarantee elitism on exact fitnesses. Since the trainer population is only a subset of the solution population, a perfect predictor can only ensure that the trainers are ranked accurately relative to each other and has no oversight in how the trainers rank relative to the solution population. However, this strategy does bias the trainer

population to ensure a diversity that both high rank variance solutions and high exact fitness solutions are well represented which may provide an edge in the emergent coevolution behavior.

2.1.3 Experimental setup

This section describes the experimental methods used to evaluate and compare the four trainer strategies. A large, randomly generated test suite on the popular symbolic regression problem is used as the basis of comparison.

Symbolic regression

Symbolic regression is a type of genetic programming optimization problem that searches the realm of symbolic expressions to minimize a specific error metric over a given data set [104]. It is an important and ubiquitous problem in genetic programming with many applications in prediction, modeling and system identification. Recent advancements in this field have extended the capabilities of symbolic regression to finding conserved, physical laws of nature [173] and inferring differential equations in dynamical systems [17]. Symbolic regression is also a good test platform because the problem difficulty is readily varied by continuous parameters such as dimensionality of the data and complexity in the problem size.

Symbolic regression is a regression problem similar to the example discussed in the **Rank prediction example** subsection. However, rather than assuming an expression and optimizing parameters within that equation, the underlying structure of the expression is a primary component of the search problem. By using data structures that can

represent any arbitrary expression, the regression problem includes finding a symbolic representation of the expression that best represents the data set. This problem is useful as it allows regression to be applied to problems where no prior assumptions can be made regarding the structure of the data source.

For a general overview of symbolic regression and how expressions are represented, refer to Section 1.3.2. The output behavior of a candidate solution is determined by resolving the expression computationally. Provided with a collection of inputs from the data set, the expression is evaluated at each point by substituting the inputs and computing each subexpression iteratively. The output can then be inserted into any standard measure, such as absolute error or correlation measures, to determine the solution fitness.

Symbolic regression was chosen as test platform for predictor coevolution for two reasons. First, the majority of fitness metrics are often local measures, which are often computed as sums of evaluations on individual points. The local independence in the fitness metrics makes selecting predictor subsamples trivial – this is not the case for problems that require fitnesses based on time series or other holistic measures. Next, there is a significant body of work regarding the development of coevolution for symbolic regression, which provides a basis of comparison [172, 176].

Randomly generated test suite

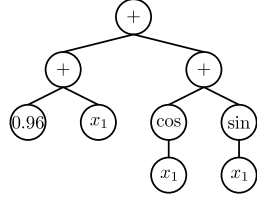
The trainer selection strategies were evaluated and compared on 256 data sets of various difficulties. To provide a fair comparison, random expressions were generated based on parameters that were swept uniformly in difficulty. The expression was then evaluated on random input data to complete the construction of the data set.

1. Select expression parameters

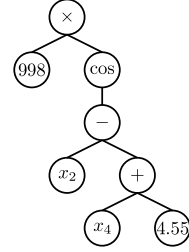
Variables: 1
Complexity: 9

Variables: 6
Complexity: 8

2. Generate and simplify tree expressions



$$y = x_1 + \sin(x_1) + \cos(x_1) + 0.96$$



$$y = 998 \cos(x_2 - x_4 - 4.55)$$

3. Generate training and validation data set

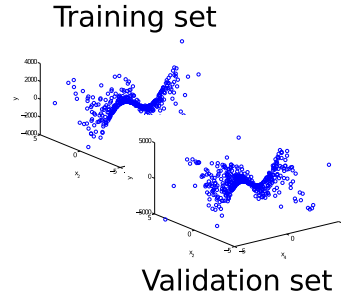
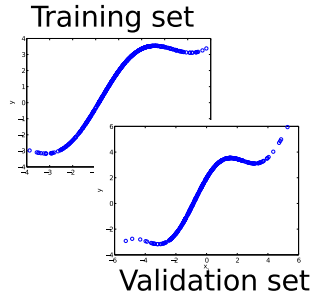


Figure 2.3: Randomly generated expressions of varying complexities.

The first step in generating random expressions is to determine the expression parameters. There are two parameters that define the difficulty of the symbolic regression problem: the dimensionality of the data, or the number of variables in the data set, and the complexity of the target function, measured as the total number of nodes in the expression's binary tree representation. For this test suite, the dimensionality of the problem was swept from 1 to 8 variables, while the expression complexity was swept from 1 to 32 nodes.

Next, a random tree was generated by continually adding subtrees, using any of the available variables, until the target complexity was met. Since many expressions have a compressible form, a symbolic simplification was required to ensure that the measure of difficulty is accurate and consistent. For example, the expression $y = 8.4 + x_2 - 2.3 -$

$x_2 + 1.7x_1$ can be simplified to lowest terms as $y = 6.1 + 1.7x_1$, resulting in a complexity of four nodes and dimensionality two variables (although x_2 is not in the expression, it is still part of data set and thus part of the search problem).

The final step is to randomly generate data points. A training data set for each expression set was constructed of 512 points. Each variable was sampled from an independent Gaussian distribution centered around the origin with a standard deviation of two. The expression was evaluated on the inputs and the resulting value was recorded as the target output. Next, a separate validation data set was also generated, consisting of 512 points as well. This data set is computed in the same fashion as the training data set, with the exception that the standard deviation of the Gaussian distributions was expanded to three. By allowing for a broader range of inputs, the validation set tests whether solutions are able to extrapolate into unseen data.

In summary, the data set consisted of 256 randomly generated expressions ranging from 1 to 8 variables and 1 to 32 nodes. Each data set has a training and validation set of 512 points, generated from Gaussian distributions of two and three standard deviations, respectively. The data was generated without noise for convenience and ease of analysis; however, previous work indicates that symbolic regression is capable of inferring models from noisy data [173, 17]. Furthermore, as the trainer population is relatively isolated from the data, the effect of noisy data on the trainer strategy is expected to be minimal. This entire process is shown in Fig. 2.3.

Experimental parameters

For the symbolic regression problem, the following evolutionary parameters were used. Mutation and crossover was done stochastically with a 1% mutation probability and

50% crossover probability. Point mutations were used and crossover was executed at a single point which was determined stochastically. Selection was done via the deterministic crowding method, which was used to minimize premature convergence on local optima [124]. A greedy, hill-climbing subalgorithm was used to optimize constant parameter values.

Expressions used a binary tree encoding with a maximum size of 64 nodes, which was chosen as the limit of human-interpretable expressions or equations that fit on a single piece of paper. The operation set contained six types: addition, subtraction, multiplication, division, sine and cosine operations. With tree sizes of 64 nodes, six operators, eight variables and parameter constants, the symbolic regression is effectively searching an expression space on the scale of approximately 10^{27} binary trees, in addition to the infinite parameter space of real-valued constants.

The solution population consisted of 128 individuals. The predictor population had eight individuals and each predictor was a subset of 8 points from the training data set. The trainer population consisted of 8 solutions and was updated every 100 generations. All of the experimental parameters were chosen based on previous work [176], and exhaustive parameter optimization was not conducted.

Implementation details

Each trainer selection strategy was evaluated on the data set of 256 expressions in two distinct runs, resulting in a total of 512 experiments for each strategy. The experiments used the same code base, with the exception of the strategies, which were swapped accordingly. Furthermore, to ensure consistency between the strategies within each run, the random number generators in the coevolved system were seeded with the same se-

ries of predetermined values. This ensured that they all began with the same random initial populations and equivalent results are obtained if the strategies made the same decisions. Thus, the coevolution algorithm was still stochastically different between runs, but identical between strategies in the same run.

Two metrics are of interest: exact fitness and convergence percentage. The fitness metric used in the problem was normalized mean absolute error (Eq. 2.6). Normalized fitness was used since it allowed comparisons between different data sets and simplified the detection of convergence. Convergence is defined as the algorithm finding the exact target expression. Since the data sets were generated without noise, the algorithm was considered to have found the exact target expression if any of the solutions computed an exact fitness of less than 10^{-3} normalized mean absolute error on the validation data set. The evolved solutions were subsequently confirmed to be symbolically identical to the target expression.

Throughout the search, the best solutions were logged, along with the exact fitness on the training and validation set, its size and the total computation effort. The computation effort is measured as the total number of equation evaluations performed in both fitness calculations. In all figures and discussions, the exact fitness reported is computed on the validation data set. The evolutionary algorithm was stopped if convergence was detected; otherwise, the algorithm was stopped if it reached 10^{10} evaluations, which is equivalent to approximately 2.5 million generations.

2.1.4 Experimental results and discussion

First, the fitness of the best solution is measured with respect to the computational effort for each trainer strategy (Fig. 2.4). All the strategies have the similar trends, suggesting

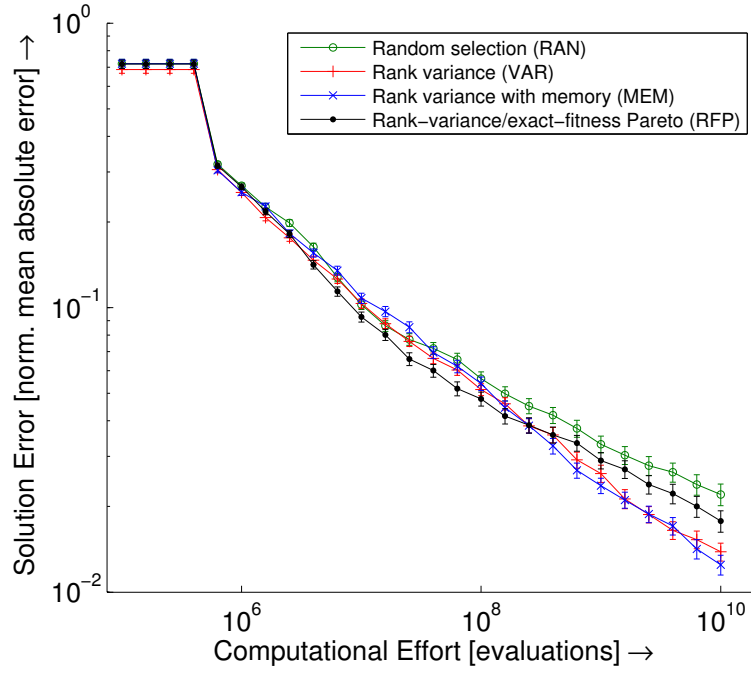


Figure 2.4: Normalized mean absolute error of the trainer selection strategies averaged across the test suite versus computational error of the coevolution algorithm. The error bars indicate standard error of the mean ($n = 512$).

that each solution population evolves in a comparable manner. Due to the quantization of trainer updates, the fitness of the best solution does not increase until after the first update in the trainer population, which occurs around $10^{5.8}$ evaluations. After the initial trainer update, the quantization effects become less significant and the error decreases in a logarithmic fashion.

Although the trends are similar, there are important differences in the overall performance of the trainer strategies. As a baseline, *RAN* achieves the worst performance. The two strategies, *VAR* and *MEM*, behave in nearly identical fashions and achieves the best results by 10^{10} evaluations. The *RFP* strategy outperforms all the other strategies at the beginning of the experiment, but ends up inferior to the *VAR* and *MEM* strategies as the computational effort increases.

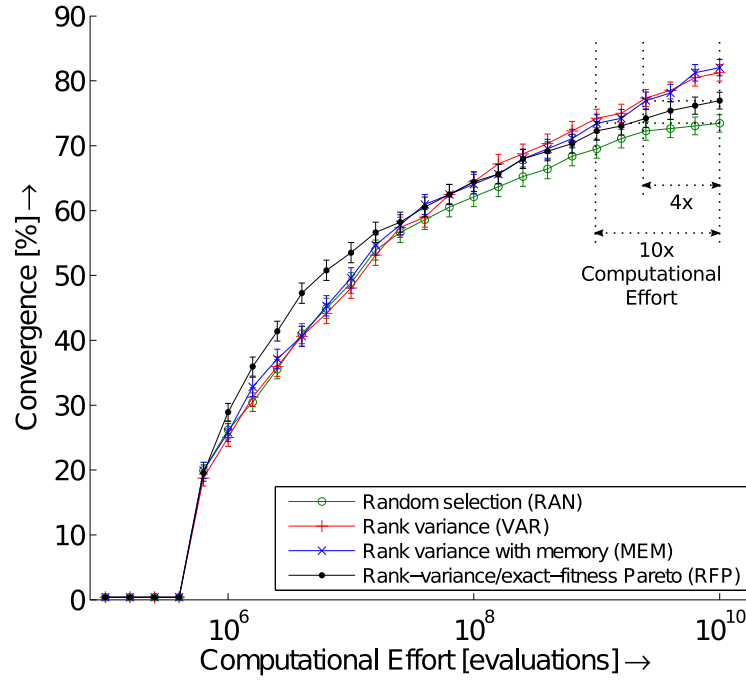


Figure 2.5: Convergence rate of the trainer selection strategies averaged across the test suite versus computational error of the coevolution algorithm. The error bars indicate standard error of the mean ($n = 512$).

A complementary trend is also observed when comparing convergence to computational effort between the trainer selection strategies (Fig. 2.5). The *RAN* strategy performs the worst, while the *RFP* strategy takes an early lead but is ultimately inferior to the *VAR* and *MEM* strategies. In fact, the *VAR* and *MEM* strategies are able to achieve the same final convergence percentage four to ten times faster than the *RFP* and *RAN* fitnesses, respectively. Also, it should be noted that every trainer selection strategy converged on the true solution orders of magnitude faster than the solitary evolutionary algorithm without predictor coevolution and is not shown.

The final performance of the strategies, with respect to the convergence and fitness metrics, are summarized in Fig. 2.6. The difference in performance is clear – the *VAR* and *MEM* strategies achieves a 1.5 times better failure rate than the baseline *RAN* strategy, while the *RFP* strategy fails 1.15 times less frequently. Similarly, the *VAR* and

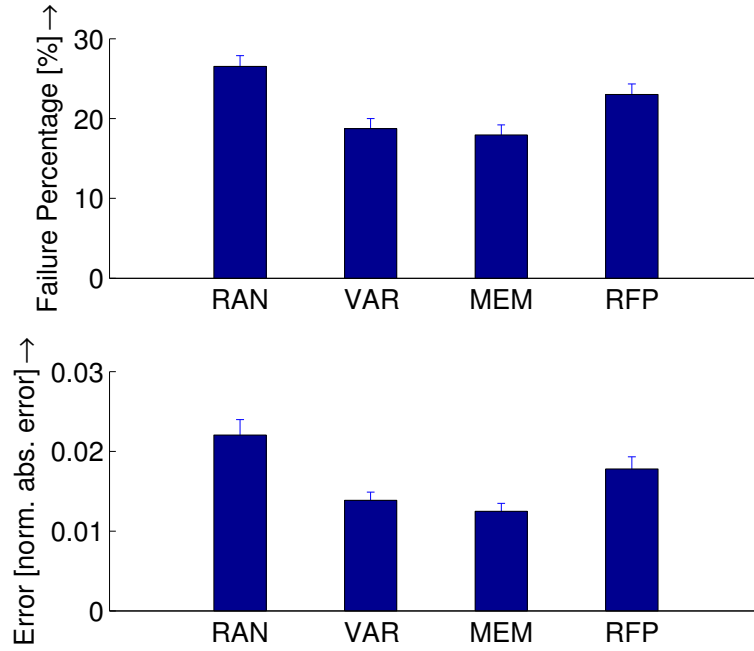


Figure 2.6: The failure percentage and solution error for the strategies at 10^{10} evaluations. The error bars indicate standard error of the mean ($n = 512$).

MEM strategies obtain 1.75 times less error than the *RAN* strategy, while the *RFP* strategy provides a 1.25 fold decrease in error.

These results are surprising and indicate that there is unexpected, emergent behavior from the tightly coupled populations. First, by comparing the *RAN* strategy with the remaining strategies, it is clear that intelligent subsampling does perform significantly better than random selection as the *RAN* strategy consistently achieves the worst results.

Next, the *VAR* and *MEM* strategies obtains the best performance by the end of the experiment in both fitness and convergence metrics. This suggests that selecting for solutions based on rank variance is a meaningful strategy as it allows the predictors to fully exploit the competitive dynamics in the coevolutionary algorithm.

Comparing the *VAR* and *MEM* strategies with each other suggest that memory does not play an important role in performance. The results imply that updating the entire

trainer population in a single step or in a serial fashion does not have a significant effect on the emergent behavior of the system. As a consequence, the *MEM* strategy is suggested as it only requires a single search for the solution of highest rank variance, while the *VAR* strategy requires multiple searches to update and fill the entire trainer population, resulting in a slightly larger overhead that is not reflected computational metric of fitness evaluations.

However, there are two parameters which could create discernible differences between the *VAR* and *MEM* strategies. The first parameter is the total run time – the performance of the two strategies might diverge if the termination condition was extended beyond 10^{10} evaluations. The second parameter is the trainer update period – this parameter was set to 100 generations for convenience and further characterization with respect to this parameter was not conducted. Changing these parameter might tease out some intricacies in the emergent, coevolutionary behavior and is left for future work.

Finally, the *RFP* strategy provides an interesting point of analysis when compared to the *VAR* and *MEM* strategies. The *RFP* is able to initially achieve superior performance but is unable to maintain its advantage as the computation effort increased. These results suggest that the *RFP* is more likely to convergence prematurely and get trapped in local optima. By adding selective pressures to model high exact fitnesses, the coevolutionary algorithm is too occupied with exploitation and does not conduct sufficient exploration.

Conversely, the results seems to suggest that elitism on exact fitnesses is not essential, but can be actually relatively detrimental, for coevolutionary progress. The *VAR* and *MEM* strategies do not encourage any elitist behavior in their predictors, and as a consequence, this may allow the predictor and solution populations to bootstrap each other out of local optima. However, using exact fitnesses may not be the best method to

promote elitism in the trainer population. A strategy to combine the early exploitation of *RFP* and the long-term exploration of *VAR/MEM* is left for future work.

2.1.5 Conclusion and future work

The goal of this work was to investigate different trainer selection strategies and their effects on coevolved rank predictor systems. Previous work on predictor coevolution focused on improving the predictor performance, often leaving the trainer selection strategy as an afterthought. Four different strategies were presented, ranging from random subsampling to selecting rank-variance/exact-fitness Pareto optimal solutions.

The four trainer selection strategies were compared on a symbolic regression problem using hundreds of randomly generated test problems, varying in problem complexity and number of variables. Of the four strategies, the *highest variance* strategies, with and without memory, was shown to provide superior results. It was able to achieve higher fitnesses and better convergence, identifying the true model four to ten times faster than the alternative strategies. Future work includes modeling trainer ranking using an underlying discrete probability distribution rather than computing variance, which assumes an underlying a continuous Gaussian distribution.

2.2 Optimal experiment design for coevolutionary algorithms using Shannon information

Active machine learning is a growing field of research that focuses on leveraging the ability to interact with the environment to improve and accelerate the learning process.

The traditional machine learning paradigm infers models by processing a fixed data set, which is assumed to comprehensively characterize the phenomena of interest. In contrast, an active approach is able to investigate the phenomena by tailoring its dynamic data set with new observations to guide the modeling process.

Although active learning is promising due to its ability to adapt, the fundamental challenge is to determine a method of data collection that results in superior models. Fine parameter sweeps in the input space may capture the required details for model inference, but this approach generates vast amounts of data, most of which is highly redundant. Furthermore, in many physical fields, measuring a single data point may be costly and difficult, making it unfeasible to gather sufficiently comprehensive data sets. Thus, collecting just the right data needed to best advance the modeling task remains a challenge.

Optimal experiment design aims to provide a solution to this fundamental issue – new data is gathered according to its ability to create disagreement between arbitrary models. By investigating areas of predicted ambiguity, the information content in each observation is maximized and superior models can be distinguished based on these observations. Furthermore, a non-parametric approach provides a powerful and robust tool that allows the comparison of candidate models without requiring analysis of the model’s underlying structure.

A popular approach to implementing active learning is through the use of teacher-learner type coevolutionary algorithms. Rather than treat active learning as a monolithic problem, it is divided into two complementary subproblems: one population infers models using a dynamic data set while the second population adds to the data set by designing experiments that disambiguates the current candidate models. Each evolutionary algorithm leverages the advancements in its counterpart to achieve superior results in a

unified active learning framework. Flavors of this coevolution active learning has been used in fields such as robotics [18] and automated science [17, 173].

This work presents a policy for optimal experiment design in active learning algorithms based on a Shannon information criterion. The problem considered is as follows: given a collection of arbitrary and competing models, determine the next experiment that will best disambiguate the models based solely on their predictions. Shannon information is used as the underlying theory for the optimal experiment criterion – traditional entropy is shown to be prohibitively computationally expensive and thus, the closely related measure *surprisal of the mean* is proposed as the optimization criterion. This approach is applied to an iterative, coevolutionary problem that uses symbolic regression for model inference and a genetic algorithm for experiment design. Complex, symbolic expressions are reliably inferred using fewer than 32 data points. Furthermore, the surprisal of the mean policy outperforms all published baselines and requires 21% fewer experiments to achieve the same performance. The surprisal of the mean policy is found to be particularly effective in situations of noisy data, local information content as well as high dimensional systems. Finally, in a real-world setting to model concrete compression strength, the surprisal of the mean policy is able to achieve 96.1% of the performance of a passive machine learning baseline with only 16.6% of the data.

2.2.1 Related work

Although coevolution has been used to primarily investigate game-theoretic constructions, there is a growing interest in leveraging the coevolutionary dynamics for active learning applications. In particular, the Estimation and Exploration Algorithm (EEA) has shown impressive results of applications in coevolutionary active learning. The

algorithm is divided into two phases: the estimation phase evolves predictive models while the exploration phase evolves experiments to gain more information. Through active learning, EEA has successfully inferred complex models using minimal trials, with applications that include inferring genetic networks [13], deterministic finite automata [14], non-linear systems [16, 17] and evolutionary robotics [18].

In addition to EEA, flavors of coevolutionary active learning have been applied to a wide variety of problems, including active sub-sampling for unbalanced data sets in GP classification [48], autonomous robot navigation [10], automatic system identification [102], coevolutionary evaluation [43] and coevolving fitness predictors [119, 172, 176]. Furthermore, coevolutionary active learning is often found in multi-agent systems; a survey that includes summaries of numerous applications can be found in [148]. However, the previous work on coevolutionary active learning often focused on the evolutionary dynamics and used an experiment design optimization criterion that was based on heuristics or intuition. In comparison, this work focuses exclusively on the optimization criterion and develops a formal framework based on information theory for experiment design.

Within the machine learning community, active learning has also been an area of growing research. As recent surveys indicate, the vast majority of active learning is focused on classification problems and recommender systems [167, 180]. The primary problem of interest is, given a collection of unlabeled examples and the ability to request labels for specific examples, how can classifiers or recommenders be improved by selective labeling [8, 24, 34]. Although this field shares many information theory concepts, there is a fundamental difference in the subject – namely, classification or recommendation instead of regression.

Cohn et al. introduced influential work on active learning for regression [35]. This work described two statistically-based learning architectures based on mixtures of Gaussians and locally weighted regression. However, their work used function approximations in the form of one of the two statistical models, as opposed to leveraging competing models simultaneously.

Gaussian processes has also found a niche is a similar capacity in active learning for regression [44, 73]. A Gaussian process model is fitted, and areas of high variance or mutual information are used to collect data. However, Gaussian processes are dictated by a single kernel or covariance function, and thus, are a single stochastic model with hyperparameters.

Finally, the concept of optimal experiment design has been rigorously explored in the control systems and system identification community [65]. The work in this field is relatively comprehensive and includes topics such as parameterized Bayesian criterion [178], uncertainties in parameter estimation [51], the correspondence with optimal control [61], least costly control identification [12] and robust system identification [164]. However, the control systems community generally begins with a model of the fundamental dynamics obtained from first principles and, consequently, are primarily focused on obtaining optimal estimates of unknown parameters in their models.

2.2.2 Optimal experiment design via information theory

Active learning consists of three, iterative processes (summarized in Fig. 2.7):

1. Model inference: the traditional learning process of inferring a model given a data set

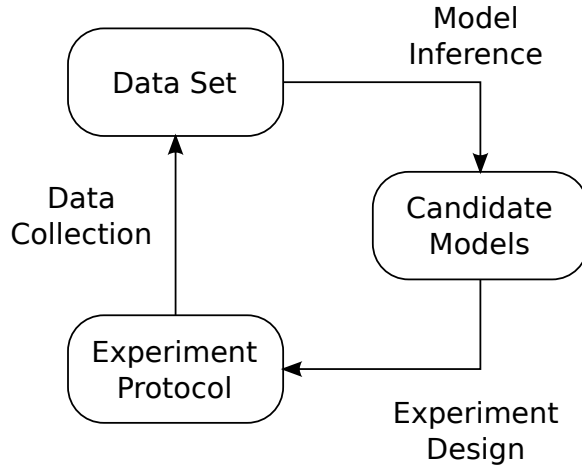


Figure 2.7: A flowchart of describing the relationship between the three processes of active learning.

2. Experiment design: the process of determining the next experiment
3. Data collection: the process of executing a desired experiment and inserting the measured results into the data set

This work presents a framework for experiment design according to a Shannon information criterion for a collection of arbitrary and competing models. This section begins with a formal problem definition and an introduction to the statistical framework. A discussion of entropy follows, along with the description of proposed criterion: surprisal of the mean. This section concludes with specific formulation for Gaussian models.

Problem definition for experiment design

Consider the following problem: given a data set and a collection of arbitrary models, determine the experiment that best disambiguates the candidate models. A distinguishing feature of this problem is that the form of the underlying system is not known. Consequently, all models are plausible and a primary goal is to build a data set that aids in inferring which candidate model best represents the underlying system.

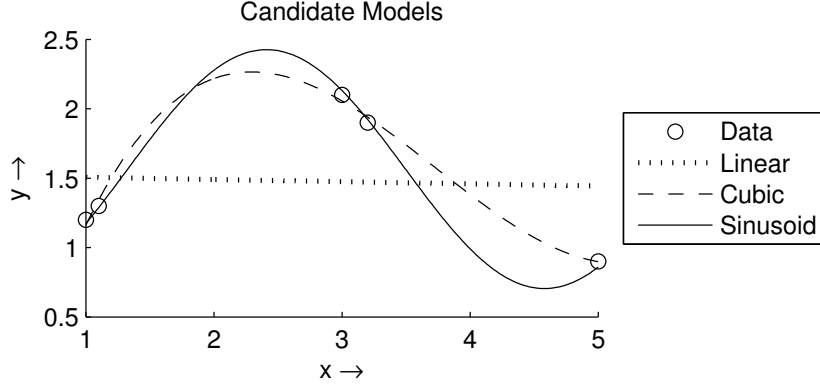


Figure 2.8: A model inference problem example with five data points and three candidate models. The goal of experiment design is to determine the optimal input, x^* , to disambiguate the candidate models and accelerate the inference process.

Consider the scenario depicted in Fig. 2.8: for a data set with five data points, there are three candidate models – a linear, cubic and sinusoid model fit with least squares regression. These models vary in their predictive accuracy, and more importantly, produce different predictions in depending on the input. Thus, the problem is selecting the next input experiment, x^* , that optimally determines which of the three models best represents the underlying system.

Note, while this example uses a relatively simple collection of models, more complex models such as feedforward neural networks or support vector regression can be applied. The distinguishing feature of non-parametric approaches is that they are agnostic to the parameter space of the candidate models and only rely on model predictions for experiment design.

Formally, for a set of multi-dimensional inputs, $x_n \in \mathbb{R}^m$, and corresponding output, $y_n \in \mathbb{R}$, consider the underlying model, $\mathcal{F}(x)$:

$$y_n = \mathcal{F}(x_n) + \epsilon \quad (2.10)$$

where ϵ is an additive noise term. The noise can be drawn from any distribution, $p(y|\mu, \theta)$, where μ is a location parameter and θ represents all the remaining model parameters.

The fundamental goal of learning algorithms is to infer the underlying model, $\mathcal{F}(x)$, from the collection of data, (x_n, y_n) . Active learning is concerned with the complementary task of determining the next input experiment, x^* , that disambiguates which candidate best represents the underlying system, given a variety of candidate regression models, $f_i(x)$.

Statistical representation of regression models

This subsection introduces a statistical framework for a collection of regression models. Rather than interpreting noise as an additive term, a candidate regression model, $f_i(x)$, can be represented statistically as a probability distribution of observing an output where the location parameter is the regression model. Thus, the regression problem (Eq.2.10) can be equivalently restated as the probability density of observing the random variable Y_i as output, y_n , given the i -th regression model, $f_i(x_n)$:

$$p_i(y_n) = p(y_n|f_i(x_n), \theta) \quad (2.11)$$

As a collection of regression models are presented, the collective predictions of the candidate models are determined via joint probabilities. Since the candidate regression models are independent by definition, the probability density of all the models observing the same output is simply:

$$p(y_n) = \prod_i p_i(y_n|f_i(x_n), \theta) \quad (2.12)$$

Finally, each candidate regression model, $f_i(x)$ is assumed to locally optimal, which is obtained by optimizing the corresponding maximum likelihood estimator (Eq. 2.13), or maximum log-likelihood estimator (Eq. 2.14), for a parametric class, $\check{f}_i(x)$:

$$f_i(x) = \operatorname{argmax}_{\check{f}_i(x)} \frac{1}{N} \sum_n \mathcal{L}(\check{f}_i(x_n)|y_n, \theta) \quad (2.13)$$

$$= \operatorname{argmax}_{\check{f}_i(x)} \frac{1}{N} \sum_n \sum_i \log p_i(y_n|\check{f}_i(x_n), \theta) \quad (2.14)$$

Note that achieving optimality for the candidate models is part of the model inference process – the experiment design process is not concerned with how the models are obtained. Approaches such as least squared regression, backpropagation or heuristic methods are all valid as long as the candidate models are locally optimal. Furthermore, optimality is only required within the same parametric class. For example, linear models are not compared against sinusoidal models.

Shannon entropy

Ideally, an optimality criterion for experiment design is Shannon entropy: a quantified measure of the uncertainty associated with a random variable. Conceptually, entropy measures the uncertainty of the collection of models – an experiment with high entropy is one where the candidate models collectively disagree while a low entropy experiment is one that is well modeled by all the candidates. Furthermore, the use of entropy is strongly motivated by related work [24, 73, 164, 178].

Before discussing entropy, the concept of surprisal or self-information of a random variable, $I(Y)$, must be introduced. Surprisal measures the information content associated with the outcome of a random variable – the rarer the event, the more information

or surprisal is obtained from observing the event. The surprisal of a random variable is:

$$I(Y) = -\log p(y) \quad (2.15)$$

Entropy, H , is then defined as the expected value of surprisal:

$$H(Y) = \mathbb{E} [I(Y)] \quad (2.16)$$

$$= - \int_{\mathbb{Y}} p(y) \log p(y) dy \quad (2.17)$$

where Eq. 2.16 is the general definition of entropy, Eq. 2.17 is the definition of differential entropy for continuous random variables and \mathbb{Y} is the support of Y . Unlike its discrete counterpart, differential entropy is not invariant to a change of variables and it can be negative. However, for experimental design, only relative entropy is relevant as entropy is used a optimality criterion and thus these properties are extraneous.

For the collection of models (Eq. 2.12), the input experiment, x_H^* , that satisfies the entropy optimality criterion is:

$$x_H^* = \underset{x}{\operatorname{argmax}} [H(Y)] \quad (2.18)$$

$$\begin{aligned} &= \underset{x}{\operatorname{argmax}} \left[- \int_{\mathbb{Y}} p(y) \log p(y) dy \right] \\ &= \underset{x}{\operatorname{argmax}} \left[- \int_{\mathbb{Y}} \left(\prod_i p(y|f_i(x), \theta) \right) \left(\sum_i \log p(y|f_i(x), \theta) \right) dy \right] \end{aligned} \quad (2.19)$$

For the majority of useful distributions, Shannon entropy for a collection of regression models is not a closed form expression. As a result, computing entropy values relies on numeric integration, which itself includes sum and product terms that scales according to the number of candidate models. Calculating entropy becomes extremely computationally expensive as the sum and product terms increase linearly with the number of candidate models, while the integration domain increases exponentially based on

the dimensionality of the input space. Furthermore, the entropy of a collection of regression models is generally non-convex and optimizing such criteria often requires numerous evaluations, compounding the computational load.

Surprisal of the mean

As Shannon entropy is prohibitively expensive to compute, a similar measure is proposed as an optimality criterion for a collection of regression models. Since entropy is defined as the expected value of surprisal or the mean surprisal, the *surprisal of the mean*, $\check{H}(Y)$, is presented as the optimality criterion:

$$\begin{aligned}\check{H}(Y) &= I(\mathbb{E}[Y]) \\ &= -\log p(\mathbb{E}[Y])\end{aligned}\tag{2.20}$$

The subsequent optimality criterion, x_H^* , is:

$$\begin{aligned}x_H^* &= \operatorname{argmax}_x [-\log p(\mathbb{E}[Y])] \\ &= \operatorname{argmax}_x \left[-\sum_i \log p(\mathbb{E}[Y]|f_i(x), \theta) \right]\end{aligned}\tag{2.21}$$

While the surprisal of the mean, referenced herein as simply *surprisal*, may not be same as Shannon entropy, it does provide a coarse approximation that is well-suited for optimization. In fact, as an optimization criterion, the two measures are closely related and there is a bounded error between the optimal values of each measure (Theorem 2.2.1). In addition to the bounded relationship to entropy, surprisal's equally important property is that it has a simple, closed-form expression that is computationally lightweight – no integration is required, the number of calculations does not depend on the dimensionality of the input space and it scales linearly with the number of candidate models.

Theorem 2.2.1. *For a bounded domain $x \in [a, b]$ and a logarithmically concave distribution $p(x)$, if x_H^* satisfies the surprisal optimality criterion, $\check{H}(x_H^*) \geq \check{H}(x), \forall x \in [a, b]$, then x_H^* is also the optimal Shannon entropy within a bounded error, $H(x_H^*) \geq H(x) - T_f(a, b), \forall x \in [a, b]$, where:*

$$T_f(a, b) = \max_{\lambda \in [0, 1]} [-\lambda \log p(a) - (1 - \lambda) \log p(b) + \log p(\lambda a + (1 - \lambda)b)] \quad (2.22)$$

Proof. For logarithmically concave distributions, Jensen's inequality defines an inequality between Shannon entropy and surprisal:

$$\begin{aligned} -\log p(\mathbb{E}[x]) &\leq \mathbb{E}[-\log p(x)] \\ \check{H}(x) &\leq H(x) \end{aligned} \quad (2.23)$$

For $x \in [a, b]$, Simic [184] provides a global upper bounds on Jensen's inequality, providing a second inequality between Shannon entropy and surprisal:

$$\begin{aligned} -\log p(\mathbb{E}[x]) &\geq \mathbb{E}[-\log p(x)] - T_f(a, b) \\ \check{H}(x) &\geq H(x) - T_f(a, b) \end{aligned} \quad (2.24)$$

where $T_f(a, b)$ is defined in Eq. 2.22.

For two different inputs, x_1 and x_2 , the optimization of surprisal is defined by the inequality $\check{H}(x_2) \leq \check{H}(x_1)$. Given the two relationships in Eq. 2.23-2.24 and the transitive properties of inequalities, the following relationship can be obtained:

$$H(x_2) - T_f(a, b) \leq \check{H}(x_2) < \check{H}(x_1) \leq H(x_1) \quad (2.25)$$

which proves Theorem 2.2.1. □

Optimal experiment design for Gaussian distributions

This subsection describes the derivation of the surprisal-based active learning framework that assumes the noise, ϵ in Eq. 2.10, is drawn from a zero-mean, Gaussian distribution with variance σ_N^2 :

$$\epsilon \sim \mathcal{N}(y|0, \sigma_N^2) = \frac{1}{\sqrt{2\pi}\sigma_N} e^{-\frac{x^2}{2\sigma_N^2}} \quad (2.26)$$

Although the Gaussian distribution is a popular and ubiquitous noise model, the surprisal framework can be readily adapted to other distributions in a similar fashion outlined in this subsection.

For a collection of regression models with Gaussian noise, the probability of observing an output (Eq. 2.12) is:

$$p(y_n) = \prod_i \mathcal{N}_i(y_n|f_i(x_n), \sigma_i^2) \quad (2.27)$$

where $\sigma_i^2 = \frac{1}{N} \sum_n (f_i(x_n) - y_n)^2$ is the empirical, sample variance of the residual error of the candidate model from the data set.

The optimal regression models are determined by optimizing the corresponding maximum likelihood estimator (Eq. 2.14), resulting in minimizing the sum of squares:

$$f_i(x) = \underset{\check{f}_i(x)}{\operatorname{argmin}} \sum_n \left(\check{f}_i(x_n) - y_n \right)^2 \quad (2.28)$$

The expected value of the joint distribution (Eq. 2.27), is the weighted mean:

$$\begin{aligned} \bar{f}(x) &= E[Y] = E \left[\prod_i \mathcal{N}_i(y_n|f_i(x), \sigma_i^2) \right] \\ &= \frac{\sum_i \frac{1}{\sigma_i^2} f_i(x)}{\sum_i \frac{1}{\sigma_i^2}} \end{aligned} \quad (2.29)$$

Thus, the surprisal optimality criterion (Eq. 2.21), is:

$$\begin{aligned} x_H^* &= \operatorname{argmax}_x \left[- \sum_i \log \mathcal{N}_i(\bar{f}(x) | f_i(x_n), \sigma_i^2) \right] \\ &= \operatorname{argmax}_x \left[\sum_i \frac{(\bar{f}(x) - f_i(x))^2}{\sigma_i^2} \right] \end{aligned} \quad (2.30)$$

The surprisal criterion for a collection of regression models with Gaussian noise is a simple, closed-form expression that is easy to calculate and computationally light. In comparison, the Shannon entropy for the same formulation (Eq. 2.19) cannot be reduced to a closed-form expression and requires numerical integration:

$$x_H^* = \operatorname{argmax}_x \left[\int_{\mathbb{Y}} \left(\prod_i \mathcal{N}_i(y_n | f_i(x), \sigma_i^2) \right) \left(\sum_i \log(\sqrt{2\pi}\sigma_i) + \frac{(\bar{f}(x) - f_i(x))^2}{\sigma_i^2} \right) dy \right] \quad (2.31)$$

For a system with m models with an input space of n dimensions and a integration grid of p intervals, calculating surprisal (Eq. 2.30) for a single data point has a computational complexity of $O(m)$. In comparison, Shannon entropy (Eq. 2.31) requires $O((2mp)^n)$ calculations.

Furthermore, it is clear that surprisal is the variance of the weighted mean, with each term weighted by the empirical variance of the corresponding model. In comparison, the standard variance optimality criterion is:

$$x_V^* = \operatorname{argmax}_x \left[\sum_i (\tilde{f}(x) - f_i(x))^2 \right] \quad (2.32)$$

where, $\tilde{f}(x) = \frac{1}{I} \sum_i f_i(x)$ is unweighted mean. This agrees with the original motivation – the optimality criterion measures how well an experiment is expected to create disambiguation between the candidate models. However, unlike standard variance, surprisal leverages their predictive abilities to emphasize disagreement between highly predictive models while marginalizing the contributions of poor candidate models.

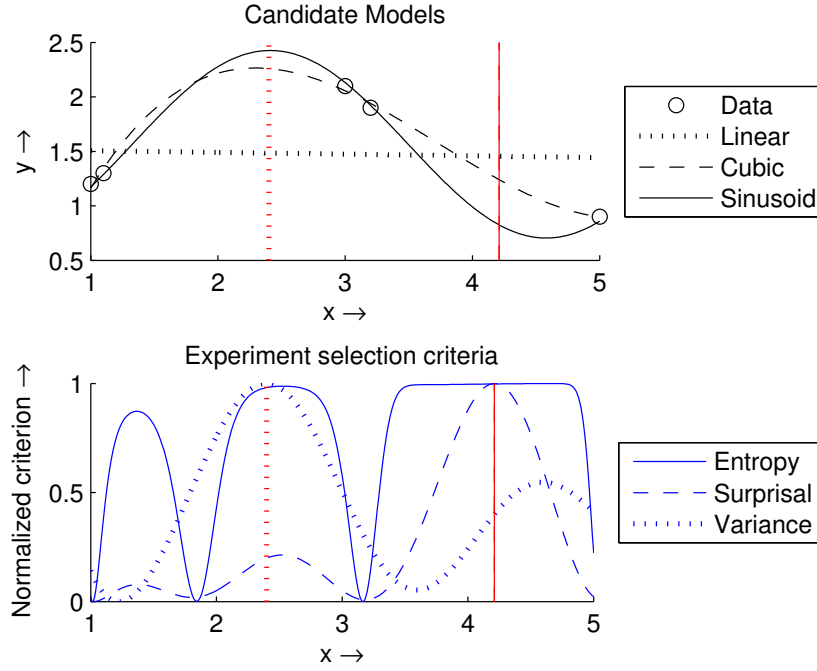


Figure 2.9: The normalized measures of entropy, surprisal and variance for the candidate model example. The optimal input experiment for each criterion is marked with red vertical curves.

Returning to the three candidate model example, the measures of entropy (Eq. 2.31), surprisal (Eq. 2.30) and variance (Eq. 2.32) are calculated and normalized within between 0 and 1 for comparison in Fig. 2.9. First, analyzing the candidate models, the cubic and sinusoid models are accurate with small residuals, while the linear model has relatively large residuals. Thus, there is more information to gain from making the cubic and sinusoid models disagree since the linear model predicts that the data is dominated by noise.

Consequently, Fig. 2.9 shows that the relative maxima of the entropy and surprisal criteria state the optimal experiment is $x^* = 4.2$, as it best differentiates whether the cubic or sinusoid model is a better representation. On the other hand, the variance criteria states that $x^* = 2.4$ is optimal as that creates the greatest disagreement between

Algorithm 2.1: Non-parametric optimal experiment design algorithm

```
1  input:  candidate models - f_i
2          data set - (x_n, y_n)
3  output: optimal experiment - x
4
5  for each model :
6      calculate empirical variance (Eq.2.27)
7
8  initialize experiment_population
9
10 until allotted computational effort is reached :
11     divide population into random pairings
12     for each pair in experiment_population :
13         recombine experiments to generate two offspring
14         mutate each offspring
15         calculate models' prediction for each offspring (Eq.2.29)
16         calculate surprisal for each offspring (Eq.2.30)
17         determine which offspring is most similar to which parent
18         for both (offspring, parent) pairs:
19             if offspring.surprisal >= parent.surprisal :
20                 replace parent with offspring
21
22 optimal_experiment = highest_surprisal(experiment_population)
23
24 return optimal_experiment
```

the predictions of all three models, with no sensitivity to the accuracy of the models. Furthermore, surprisal is qualitatively similar to entropy with three local maxima and a similar shape, while the variance curve bears little resemblance to entropy.

The non-parametric optimal experiment design process is outlined in Algorithm 2.1. The algorithm is a genetic algorithm that maximizes surprisal and uses deterministic crowding for selection [124]. Note that maximizing surprisal is the primary goal and can be obtained by applying alternative evolutionary algorithms or optimization approaches.

2.2.3 Coevolutionary active learning testbench

This section describes the testbench that used to measure the performance of the proposed experiment design within an active learning framework. The section is organized

according to the processes outlined in Fig. 2.7; model inference, experiment design and data collection. This section is concluded with a discussion of the procedure.

The coevolution aspect of this approach is a product of how the active learning processes are implemented. The model inference process is implemented as a symbolic regression algorithm that creates genetic programs to model the dynamic data set, while the experiment design process is a genetic algorithm that finds new data which optimizes the surprisal given the current models. This type of implementation produces dynamics where the performances of both evolutionary algorithms are interdependent.

Symbolic regression via genetic programming

For model inference, symbolic regression, a type of genetic programming algorithm [104], is used – a process flowchart is shown in Fig. 2.10. For additional details regarding symbolic regression, refer to Section 1.3.2. Symbolic regression was chosen for variety of reasons. First, it is a heuristic algorithm and, consequently, highly sensitive to the data set. As surprisal works with arbitrary regression models, one that is more sensitive to data sets provides a better measure of robustness.

Next, symbolic regression provides an automated method to explore various forms based on their predictive accuracy. In comparison, alternative inference algorithms focus on optimizing the parameters of a single form. As a multiple candidate models are required, manually selecting different model types creates a bias in comparing active learning algorithms. Symbolic regression is able to represent arbitrary expressions by arranging fundamental building blocks and generates new models in a heuristic fashion.

A continuously updated record of the best performing models is kept, where performance is measured according to a Pareto optimal approach based on accuracy and

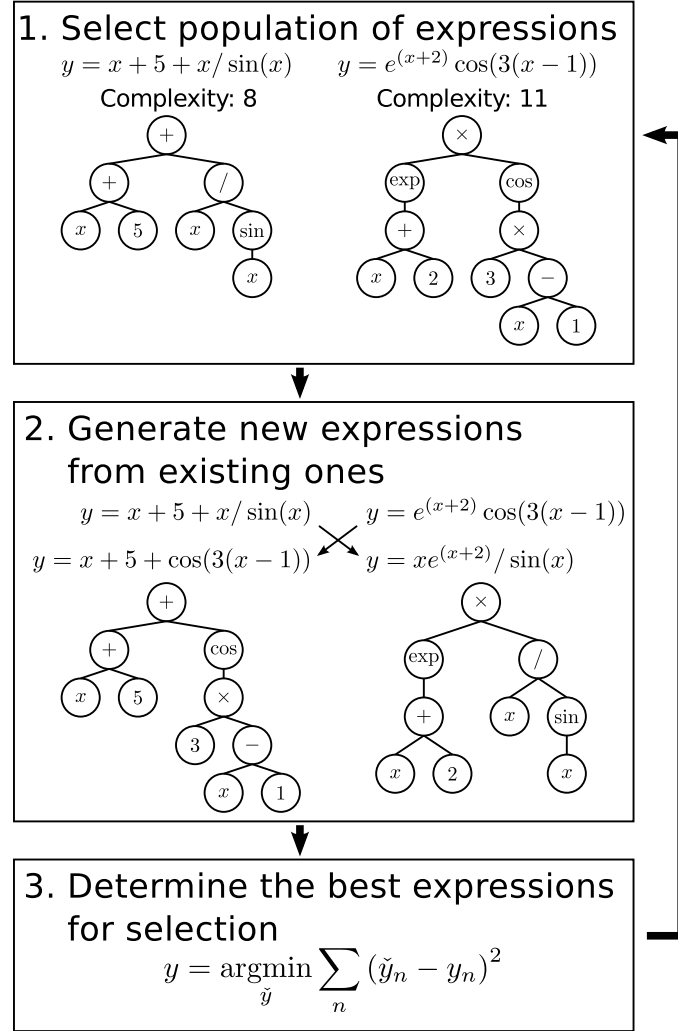


Figure 2.10: A process flowchart for symbolic regression.

complexity – for every model in the record, no other models were found that are simultaneously more accurate, as determined by Eq. 2.28, and less complex, defined as the size of the expression. These Pareto optimal expressions are used as the candidate models. For additional details regarding fitness-complexity Pareto analysis, refer to Section 1.3.4.

A publicly available symbolic regression engine, Eureqa, was used for this test-bench [177]. Symbolic regression was executed for 10^9 evaluations, a conservative

estimate of the computational effort needed for convergence, which required approximately 5 core minutes on a 2.8GHz Intel processor. The building block operators were $\{+, -, \times, /, \sin, \cos, \exp\}$ and the default parameters were used: population size of 64 solutions, crossover probability of 0.7 and mutation probability of 0.03. The performance of the algorithm was not found to be sensitive to these parameters.

Experiment design

Four experiment design policies were compared; entropy was not included due to its prohibitively long computational demands. Two policies are *model-free* and are independent of any input, while the other two policies are *model-based* and leverage the candidate models as part of the design process.

- Binary recursive design [Recursive] – This is the optimal method for a model-free design policy. The input space is recursively subdivided in half along each dimension. For 1D input with a domain $[0,1]$, the binary recursive policy generates the following sequence of experiments: $\{0, 1, \frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}, \frac{3}{8}, \frac{5}{8}, \frac{7}{8}, \frac{1}{16}, \dots\}$.
- Random design [Random] – This is a model-free design policy that consists of randomly selecting an experiment with a uniform probability.
- Variance design [Variance] – This is a model-based design policy based where the optimality criterion is Eq. 2.32. This policy was included due to its popularity in related work [35, 51, 61, 167].
- Surprisal design [Surprisal] – This is the proposed, model-based design policy where the optimality criterion is Eq. 2.30.

Experiments were selected within predefined bounds for the input space. Although the framework is suitable for continuous inputs, the experiment selection was discretized

into 257 uniformly spaced grid points per input. The discretization was chosen as a result of the performance metric discussed in the **Testbench procedure** subsection.

The model-based policies are generally non-convex which presents an optimization challenge. A simple genetic algorithm (Algorithm 2.1) was used that evolved bit strings that corresponded to the grid points. The genetic algorithm had a population size of 8, crossover probability of 0.7 and a mutation probability of 0.05. Since this optimization problem was relatively simple, the algorithm ran for 10^3 evaluations, which required approximately 5 core seconds on a 2.8GHz Intel processor.

Data collection

Three different underlying models were selected based on their difficulty for symbolic regression algorithms. The underlying models generate data according a symbolic expression and thus, a perfect model inference is possible with this approach. These underlying models were adapted from popular baselines and the difficulty of these models for symbolic regression is well established [172, 208].

1D exponential sinusoid The exponential sinusoid is a one-dimensional problem with the expression $y = e^{0.1x^2} \sin(5.5x)$ and a complexity of 11 nodes. A plot of the function evaluated at the possible 257 experiments is shown in Fig. 2.11.

This expression is designed to be a simple inference problem. It is a one dimensional problem with lower complexity than its counterparts. Furthermore, the function has *global information content* – any experiment is likely to provide meaningful information for model inference since the probability of generating a pathologically poor data set, such as one that consists of only zero crossings, is highly unlikely.

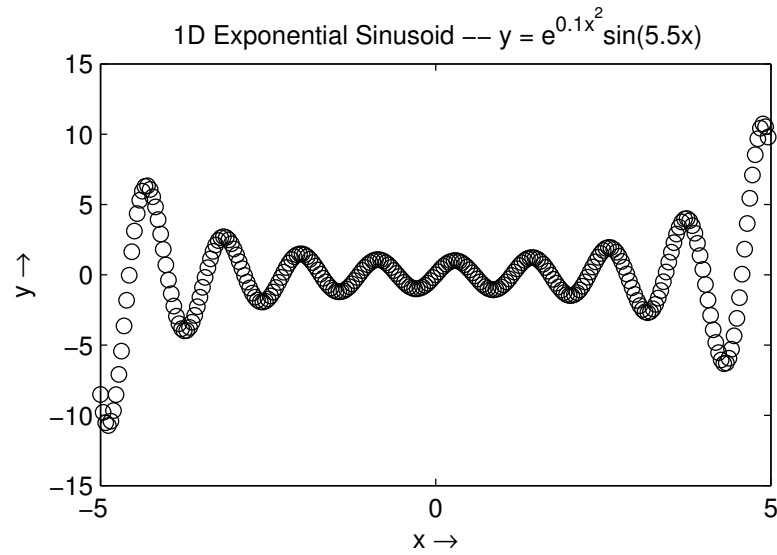


Figure 2.11: A plot of the exponential sinusoid expression.

1D wavelet The wavelet is a one-dimensional problem with the expression $y = e^{-(x-2)^2} \sin(5x)$ and a complexity of 15 nodes. A plot of the function evaluated at each of the possible 257 experiments is shown in Fig. 2.12.

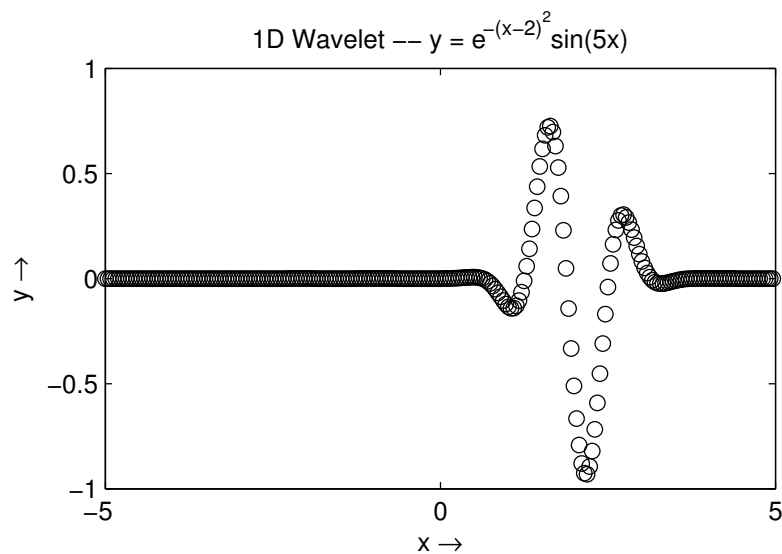


Figure 2.12: A plot of the wavelet expression.

Unlike the exponential sinusoid, the wavelet presents a significantly more challenging inference problem due to its *local information content*. The majority of the input space consists of near zero outputs, and thus it is trivial to obtain a pathologically poor data set. Much of the expression is characterized within a narrow input space, and thus a non-uniform sampling of the input space is required for efficient inference.

2D pulse The pulse is a two-dimensional problem with the expression $y = \frac{x_1 + x_2 + 0.1}{10x_1^2 + 5x_2^4 + 1}$ and a complexity of 24 nodes. A contour plot of the function evaluated at each of the possible 257×257 experiments is shown in Fig. 2.13.

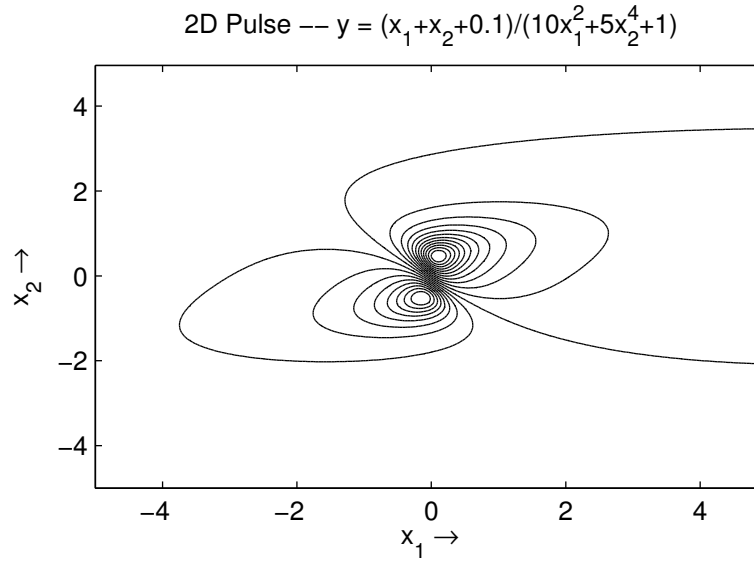


Figure 2.13: A plot of the pulse expression.

Like the wavelet, the pulse has *local information content* and presents a challenge for experiment design to avoid deceptive data sets. The pulse also presents the additional problem of inferring a multi-dimensional function – the space of possible experiments grows exponentially with the number of inputs. This phenomenon is referred to as the curse of dimensionality and is often a crippling issue in learning algorithms.

Noise Corruption – Each of the data sets were corrupted with additive, zero-mean Gaussian noise. Given that the sample variance of the total data set is σ_S^2 and the noise variance is σ_N^2 , then the signal-to-noise ratio is defined as:

$$SNR = 10 \log_{10} \left(\frac{\sigma_S}{\sigma_N} \right) \quad (2.33)$$

Three treatments were considered: $\sigma_N^2 = 0$, or a noiseless output; $SNR = 20\text{dB}$, or when the signal is 100-fold stronger than the noise; and $SNR = 10\text{dB}$, or when the signal is 10-fold stronger than the noise.

Testbench procedure

The testbench begins with three randomly-selected, initial experiments. The active learning processes, Fig. 2.7, are iteratively executed in a sequential order: first, models are inferred using the existing data set; this is followed by an experiment design using one of the four policies in the **Experimental design** subsection; and finally, the experiment evaluated and the result is inserted into the data set. This computationally intensive algorithm is motivated by the scenario that building a data set is expensive, while computation is relatively cheap.

This iterative process is repeated 32 times, resulting in a final data set of 35 points. The number of experiments was chosen to illustrate the performance difference between the various experiment design policies, and was found to be a suitable balance between the asymptotic behaviors of perfect accuracy with infinite experiments and poor accuracy with zero experiments. The active learning algorithm is applied to the three expressions with three noise conditions each, resulting in 9 treatments. Finally, for each treatment, the algorithm is repeated 12 times for statistical analysis with different initial experiments.

The performance is measured using the normalized mean squared error:

$$E = \sum_n \frac{(f(x_n) - y_n)^2}{\sigma_s^2} \quad (2.34)$$

which effectively is the normalized, discrete integrated error between the candidate model and ground truth. The normalized error is computed for all the candidate models and lowest error is reported.

2.2.4 Testbench results

This section describes the results of the active learning testbench. First, the convergence of the design policies is analyzed, followed by a case study of designed data sets.

Convergence analysis

Convergence, or the normalized error compared to the size of the data set, is shown for all treatments in Fig. 2.14. Overall, surprisal is the superior method as it converges at least as fast as any other policy for every treatment. In fact, on average, surprisal is able to achieve the same performance as the next best baseline with 6.8 fewer experiments, or a 21% reduction in the number of experiments. Furthermore, surprisal achieves significantly better convergence as the model inference problem increases in difficulty, either by increasing the noise corruption, increasing the input dimensions, or localizing the information content.

The noiseless exponential sinusoid expression was designed as the simplest, non-trivial problem for every policy as it is a one-dimensional, noiseless model with global information content. Surprisal performs just as well as any other policy, suggesting no

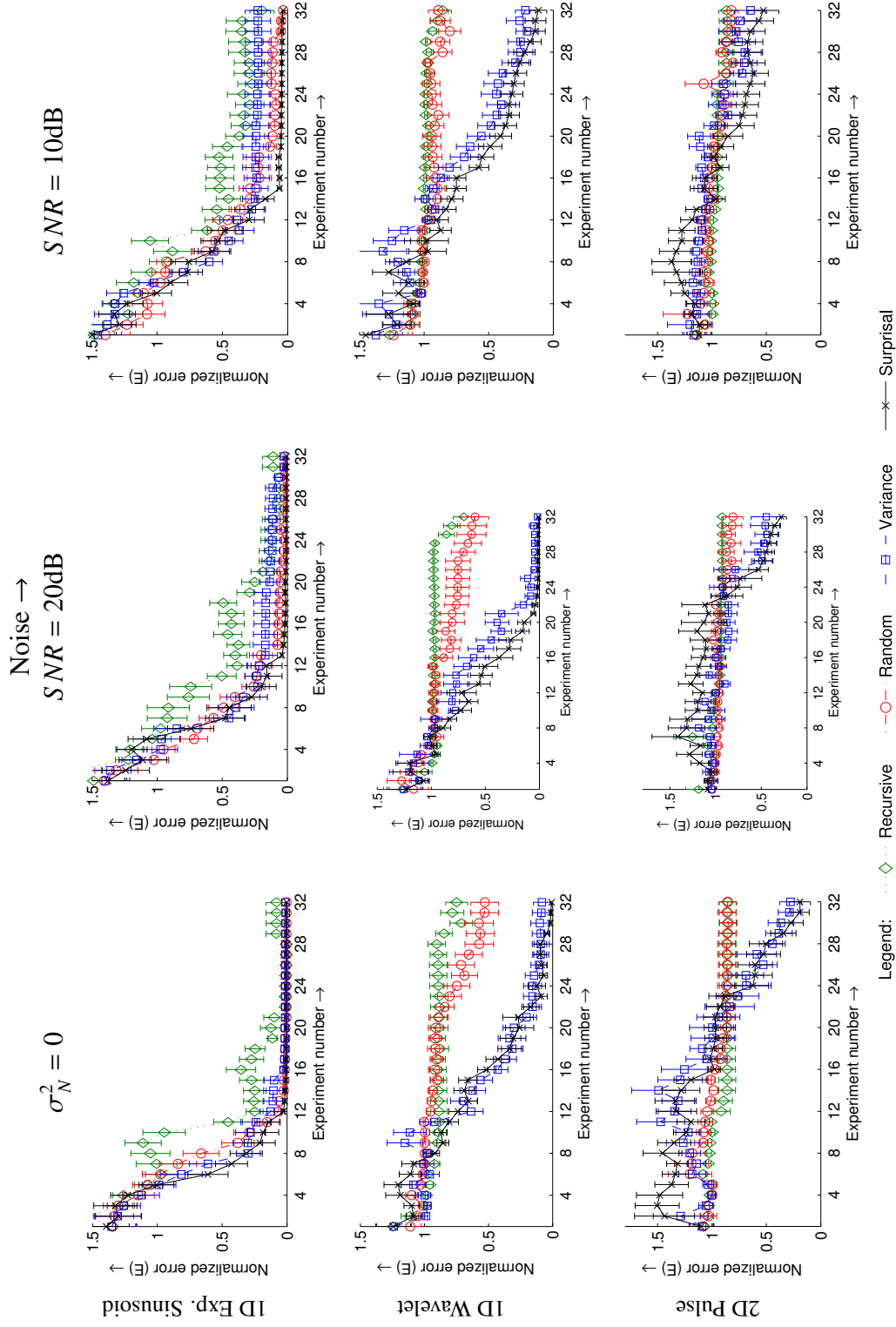


Figure 2.14: The convergence plots for the three testbench expressions and the three noise levels. Error bars indicate standard error ($n = 12$).

penalty or overhead is occurred for simple problems. As the noise levels increase for the exponential sinusoid, surprisal provides significant improvements on convergence.

The wavelet expression compares the effect of information content. The best predictions in the model-free design policies, recursive and random, achieve a normalized error of approximately 1, which is marginally better than predicting a constant mean or reducing the data to its first statistical moment. On the other hand, the model-based policies, variance and surprisal, are able to identify that the information content is not uniform and localize the data set for superior convergence. In addition, this difference is magnified with increasing noise levels.

The analysis of the pulse expression provides two key results. First, for this expression, the surprisal design policy is not severely affected the increase of dimensions. Although the convergence requires slightly more data than the one-dimensional examples, the increase in the number of experiments does not reflect the exponential growth in the input.

Next, an interesting phenomenon occurs for model-based policies on difficult inference problems: for small data sets, the normalized error suggests that policies obtain poor representations of the underlying expression. This phenomenon arises from the fundamental experiment design principle of determining experiments that cause disagreement in the candidate models. Consequently, this results in pathological sampling that promotes overfit models for small data sets.

However, in the long term perspective, this phenomenon is actually advantageous within the active learning paradigm. Overfitting occurs when an inferred model describes noise contributions instead of the underlying model, and thus, makes bad predictions with great certainty. As a consequence, overfit models are ideal for experiment

design as they create large disagreements. These disagreements are then iteratively used to build better data sets and overfit models are quickly removed due to their inability to generalize to the new data. Thus, the model-based policies are able to leverage the existence of early overfit models to accelerate the data collection process, eventually resulting in superior models.

Case study analysis

In addition to analyzing convergence, the designed data sets themselves provide a significant resource for analysis. For the wavelet problem with the highest noise condition, $SNR = 10\text{dB}$, the data set of the median performing run for each policy is shown in Fig. 2.15.

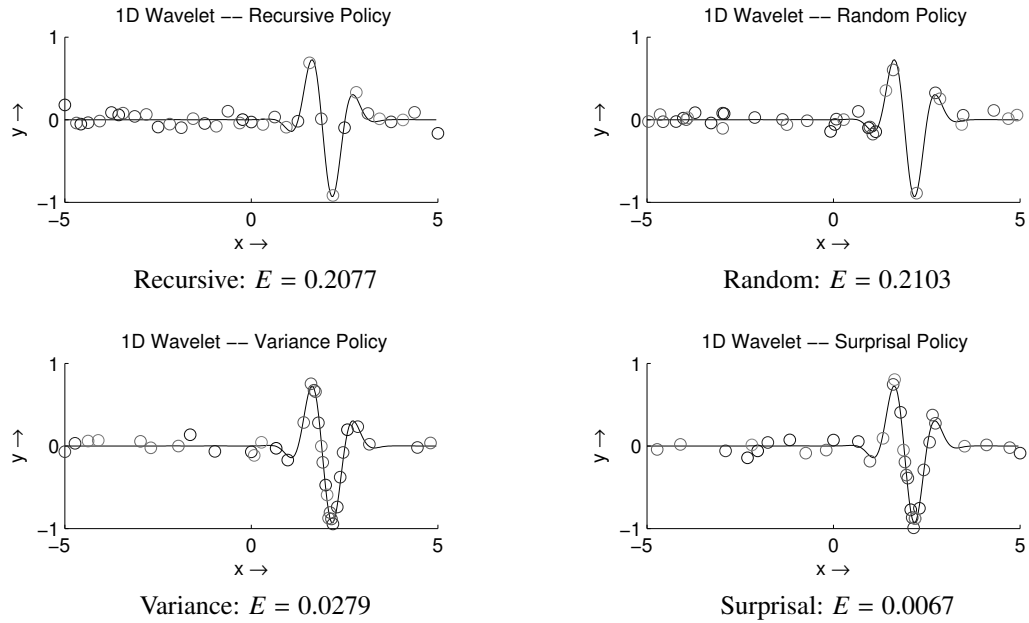


Figure 2.15: The high noise, $SNR = 10\text{dB}$, data set of the median performing run for each design policy overlaid on the ground truth expression. The final normalized error is indicated. Darker points indicate early experiments while lighter points indicate late experiments.

The two model-free policies, unsurprisingly, generate data sets with a relatively uniform distribution; they are unable to focus on the characteristic region, resulting in poor performing model inferences.

Variance is able achieve an order of magnitude reduction in error by requesting a high density of experiments in the region that helps characterize the underlying model. In this run, note the limited noise corruption on the characteristic region; this suggests that despite the increased density, variance is still highly sensitive to noise.

In comparison, surprisal is able to achieve another order of magnitude reduction in error with a data set that is severely affect by noise, even in the characteristic region. Because the surprisal design policy accounts for the model prediction accuracy, it attempts to disambiguate overfit models, which is equivalent to determining the magnitude of noise. As such, it resamples at pivotal and specific locations, allowing it to deal with greater noise conditions than the variance design policy.

2.2.5 Real-world concrete compression strength experiments

This section describes an application of coevolutionary active learning for a real-world problem. The goal is to infer a symbolic model of the concrete compression strength as a function of various mixing parameters [214], using experimental data that is publicly available in the UCI machine learning database [62]. The data set consists of 1030 measurements with 8 inputs, which describes the amount of mixture components as well as the settling time.

This example of a real-world setting illustrates the possible applications for coevolutionary active learning. For domains that are dominated by experimental testing such

as materials engineering, gathering extensive data sets for traditional machine learning algorithms may be financially prohibitive. Active learning provides an alternative approach that is able to achieve similar performance levels as without expensive parameter sweeps.

To simulate the active learning environment, the data is divided into three sets: a potential training pool, an active training set, and a test set. The original 1030 point data set was reorganized to generate four, unbiased samples: the original data set was randomly divided into four subsets and, in a cyclic fashion, one subset was designated as the test set while the remaining three formed the potential training pool. The experimental design and data collection processes then consisted of iterating through the potential training pool, finding the inputs that maximize the experiment design policy and transferring that observation to the active training set for model inference.

Three treatments were selected for comparison. The first two treatments used active learning approaches with variance and surprisal as the experiment design policy, respectively. The protocol in Section 2.2.3 was used and the initial active training set consisted of three data points selected randomly from the potential training pool. Model inference was executed via Eureka with a computational limit of 10^9 evaluations and 128 experiments were designed. The third treatment used a passive machine learning approach that consisted only of model inference using all available 773 observations as the training set. The computational effort was extended to 3.2×10^{10} evaluations to compensate for the larger training set. For added statistical analysis, each treatment was repeated eight times for each sample, resulting in 32 runs for each treatment.

The performance of the three treatments is shown in Fig. 2.16. It is clear that the surprisal experiment design policy is able to significantly outperform its variance-based counterpart. After 128 designed experiments, surprisal achieves a normalized error of

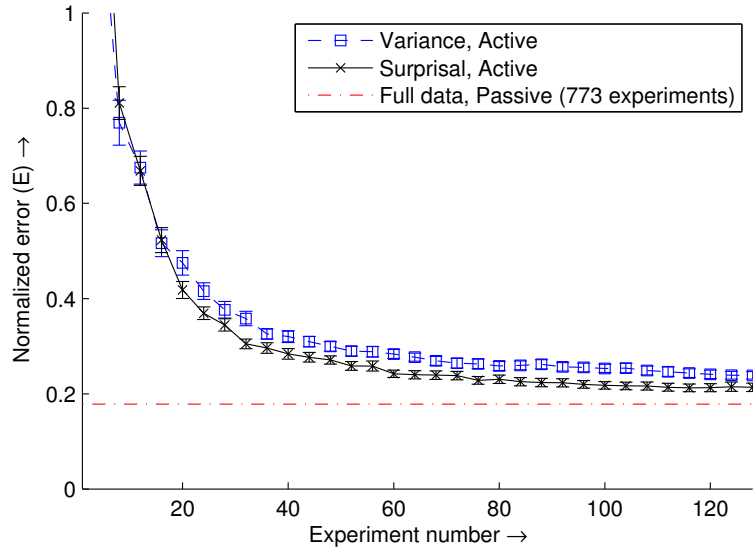


Figure 2.16: The convergence plot for the three treatments. Note that the full data treatment is a passive learning approach and uses all available data for its training set while the variance and surprisal treatments dynamically add to their training set and the number of instances is equal to the experiment number plus three. Error bars indicate standard error ($n = 32$).

$0.210 \pm .006$ while variance has an error of $0.234 \pm .007$. However, the more important metric than final error is the rate of learning – the surprisal policy is able to achieve the final error of the variance policy with just 59 experiments or with 53.8% less experiments. Surprisal is able to achieve this significant difference in performance because of its ability to account for the candidate models’ accuracy, which is essential for complex, noisy systems.

Next, the surprisal design policy is able to asymptotically approach the error of the full data treatment, $0.178 \pm .003$. In fact, it is able to achieve 96.1% of the performance with just 16.6% of the data. For systems where data collection is limited, active learning is able to approach similar levels of performance with significantly fewer experiments.

However, all of the treatments fare noticeably worse than the neural network approach in the original work [214]. The difference in performance may be a result of three sources. First, it is possible that concrete compression strength is simply better suited for modeling by feedforward neural networks as opposed to symbolic expressions. Second, the active learning approach was simulated by restricting the experiment design to experiments that had recorded observations. It is possible that the active learning would be able to find better symbolic models if it was able to obtain specific observations of its choice. Finally, many of the important mixture properties, such as the water-binder ratio, are not drastically non-linear and have global information content [215]. From Section 2.2.4, it is clear that active learning approaches, and surprisal in particular, perform best in local information content systems. Nonetheless, the surprisal learning approach does remarkably well compared to the more appropriate baseline of full data symbolic regression.

2.2.6 Conclusion and future work

In this work, *surprisal of the mean* was proposed as an optimality criterion for non-parametric experiment design for coevolutionary active learning. As a coarse approximation to entropy, surprisal provides the primary benefit of being computationally lightweight. Compared on a testbench of three different underlying models and various noise conditions, surprisal outperforms the three baselines requiring 21% fewer experiments to achieve the same performance, particularly in situations where the underlying model has local information content, high noise conditions or high dimensionality. Furthermore, surprisal performs at least as well as any baseline. The resulting data sets are able to capture the underlying model for reliable model inference with as few as 32 data points. For the real-world concrete compression strength experiments, surprisal is

able to achieve the same final error as variance with 53.8% less experiments and obtain 96.1% of the performance of the full data baseline with only 16.6% of the data. Future work includes investigating how this framework can be extended to designing a sequence of experiments.

2.3 Coevolutionary predictors for kinematic pose inference from RGBD images

A fundamental issue in a multitude of robotic and computer vision applications is the automated, three-dimensional pose inference of an articulated subject (Fig. 2.17). For example, teaching complex robotic movements via human demonstration relies on the ability to infer the teacher’s pose [47, 161]. While recent advances have made capturing three-dimensional depth images convenient and affordable, extracting pose information from these images remains a challenge.

Ideally, pose inference operates by manipulating a kinematic skeleton of the subject to best explain the depth image, which provides a natural and robust approach. However, kinematic-based pose inference has often been considered an intractable problem due to a variety of reasons [64, 183], including the density of locally optimal solutions, the high dimensional problem space of articulated kinematic structures and the computational limits of dealing with point clouds, which of thousands of points from a single image.

As a result, state of the art methods in markerless pose inference revolve around two approaches: pose recognition [64, 182] and visual hull methods [37, 63]. While these approaches are fast and accurate, they rely on extensive, supervised training with

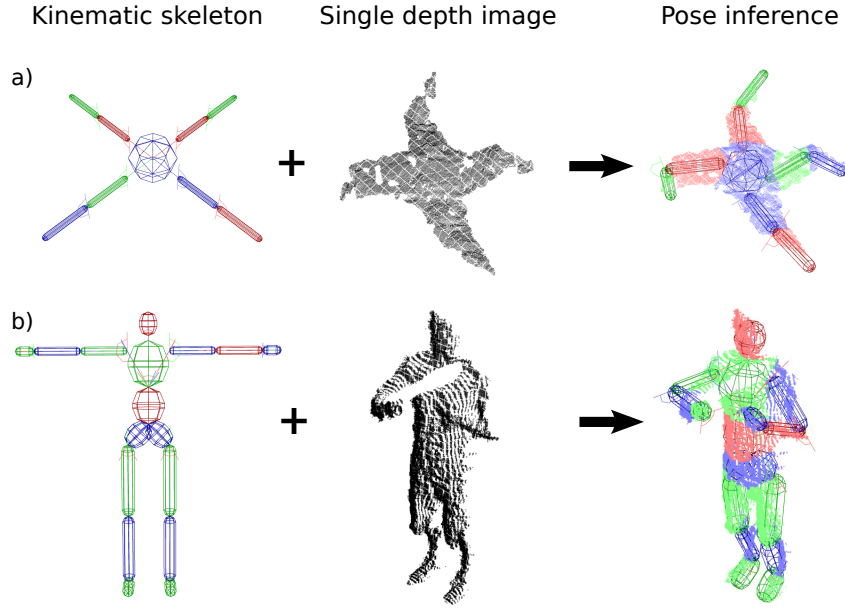


Figure 2.17: Inferring pose information from a single depth image and an arbitrary kinematic skeleton. The framework is able to pose both (a) quadrupedal spider and (b) humanoid kinematic skeletons without any modifications or training.

vast data sets, and thus, they are constrained by the composition of the data set and the lengthy training time.

Nonetheless, a method to infer poses using only the kinematic structure is still profoundly desirable as it could operate in an unsupervised manner. A co-evolutionary framework is able to overcome the traditional limitations in kinematic pose estimation by leveraging competitive interactions between two populations to provide a tractable and reliable approach. Rather than evolving poses using the whole point cloud, a second population of subsampled points are simultaneously co-evolved to disambiguate the competing poses while also significantly reducing the computational load.

This section presents a general approach to inferring poses of arbitrary kinematic skeletons from a single depth image without prior training. The pose inference problem is defined as a model-based optimization and a learning algorithm based on co-

evolutionary approaches is designed to efficiently search the vast parameter space. The primary contributions of this work include: a volumetric parameterized description of kinematic skeletons, an effective fitness metric for pose estimation, and a co-evolutionary framework for the computationally intensive inference problem. The algorithm is applied to 34 and 78 degree of freedom models and reliably infers the model parameters for image reconstruction of point clouds with over 40,000 points. The inference algorithm is shown to be robust and can even accurately infer poses with non-trivial self-occlusions.

2.3.1 Related work

The vast majority of pose inference research focused on exclusively the human kinematic skeleton. Recent surveys [135, 154] describe two primary directions: pose assembly via probabilistic detection of body parts and example-based method. Pose assembly attempts to reconstruct the pose by first identifying body parts using pairwise constraints including aspect ratio, scale, appearance, orientation and connectivity. In contrast, example-based methods compare the observed point cloud with a database of samples. A primary limitation of these techniques stems from their supervised learning foundation: inference requires *labeled* training data and the generality of the inference algorithm depends on the content of the training data.

Body part classification has been successfully adapted to accurate, real-time implementations [64, 198]. Shotton et al. described a particularly successful approach to human pose recognition that builds a probabilistic decision tree to first find an approximate pose of body parts, followed by a local optimization step [182]. While this technique is fast and reliable, it relies on significant training exclusive to the humanoid skeletal

structure: 24000 core days of training on 1 million randomized poses. The algorithm learns a prior distribution of likely poses from the training set – consequently, the algorithm will do poorly for a test point cloud that is not within the learned prior distribution and thus lacks robustness with respect to arbitrary point clouds.

Due to the limitations of training-based algorithms, there are a variety of alternative approaches under investigation, including visual hull methods, interactive kinematic inference and particle-swarm optimization based methods.

Visual hull methods are approaches that do not depend on training data [37, 100]. In these approaches, an outer hull is mapped to the kinematic skeleton in advance using human experts, reducing the task to a hull-matching problem and rather utilizing the kinematic skeleton. For example, Gall et al. used laser-scanned models to find poses of complex models generated from animals and non-rigid garments in a markerless camera system [63]. This approach requires an accurate model per subject and cannot be readily adapted to generic or unknown subjects.

Katz et al. introduced an interactive method that infers relational representations of articulated objects by tracking visual features [94]. While this work does not focus on pose inference directly, it presents a framework to extract kinematic information from an unknown object using computer vision. However, it is limited to planar objects and requires a variety of interactions with the object.

A recent development in markerless pose estimation is the introduction of particle-swarm optimization, applied to kinematic skeletons of the human upper torso [163] and humanoid hands [141]. While these approaches approached real-time implementations on GPUs and provided direct searches on the pose parameter space, the largest

demonstrated model contained 27 degrees of freedom with sparse point clouds containing around 1000 data points, a significantly simpler computational problem.

A primary application of unsupervised pose inference is teaching by demonstration, which has been shown to be an efficient and natural method to transfer knowledge to robots. Riley et al. used imitation to achieve human-like behavior in highly-complex, humanoid robots [161] while Kober et al. explored how to use demonstrations to learn motor primitives and tackle complex dynamics problem via reinforcement learning [101]. Although, this illustrates potential uses for automated pose inference in robotics, current teaching by demonstration implementations rely on predefined transformations and there have been no attempts to generalize to arbitrary teachers. Better pose inference could also help in improving performance of human activity detection [194].

2.3.2 Pose inference algorithm

Given a point cloud from the RGBD sensor, the algorithm poses a given kinematic skeleton to best explain the depth image. The volumetric parameterized representation of kinematic skeleton is first described, followed by the motivation and description of the fitness metric. This section concludes with a description of the evolutionary computation-based learning algorithm.

Kinematic models

Selecting a suitable representation of kinematic models is essential to inference, as an efficient encoding allows for generality as well as simplicity. The kinematic model is rep-

resented as a collection of rigid links, organized in an acyclic graph structure (Fig. 2.18). The root node represents a frame of reference that describes the position and orientation of the model origin. The root parameters are unbounded.

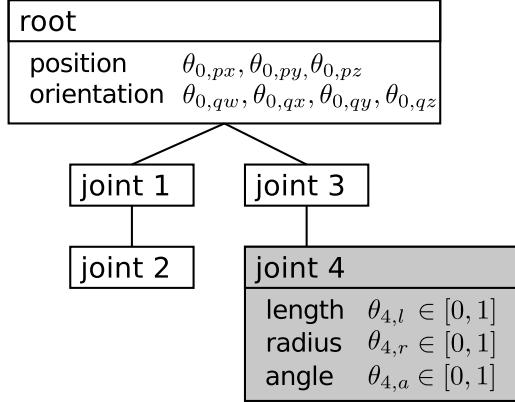
Each child in the acyclic graph represents a rigid link that is modeled as a piecewise combination of cylinders and hemispheres. Although links are traditionally represented as line segments, a volumetric representation was chosen to match the 3D information of depth images. This parameterization defines a volume that is the locus of all points that are a constant radius away from a line segment.

Each link is described by three free parameters: link length, link radius and joint angle. The model parameters are defined using a parametric equation with two predefined bounds, and linear interpolation or SLERP [181] is used accordingly. The bounds allow for anatomically consistent definitions in the kinematic model. By defining the link radius with respect to the link length, the link maintains its length regardless of the radius and interpolates between a line segment to a complete sphere. This model allows for efficient geometric computations, such as finding distances of a point to the surface or collision detection (Eq. 2.37). Although individual links only provide a single degree of freedom, complex topologies, such as ball and sockets joints, can be obtained by cascading multiple zero-length links.

Fitness metric

As with all evolutionary algorithms, a fitness metric is defined that gives higher scores when the model better explains the observed point cloud. There are two criteria in this term: self-collisions must be avoided and the observed points must be well explained.

a) Acyclic graph



b) Parametric link equations

Link length

$$l_j = l_j^{\min}(1 - \theta_{j,l}) + l_j^{\max}\theta_{j,l}$$

Link radius

$$r_j = \frac{1}{2}l_j[r_j^{\min}(1 - \theta_{j,r}) + r_j^{\max}\theta_{j,r}]$$

Joint angle

$$q_j = q_j^{\min} \frac{\sin[(1-\theta_{j,a})k]}{\sin k} + q_j^{\max} \frac{\sin \theta_{j,a}k}{\sin k}$$

where $\cos k = q_j^{\min} \cdot q_j^{\max}$

c) Kinematic model

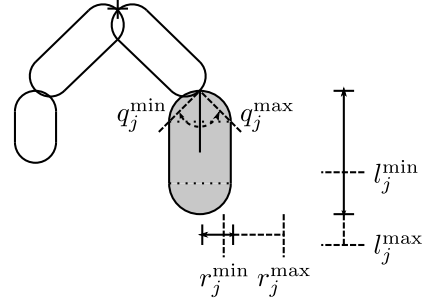


Figure 2.18: a) The acyclic graph representation of the skeleton, b) the intermediate parametric equations and c) the corresponding visual depiction. Link 4 is highlighted for reference.

Thus, the following fitness metric is proposed:

$$F(\theta) = -(1 + \epsilon c) \left[\frac{1}{N} \sum_{n=0}^N \ln \left(1 + \frac{\|\vec{p}^*(\theta) - \vec{p}_n\|}{\sigma} \right) \right] \quad (2.35)$$

where θ are the collection of model parameters depicted in Fig. 2.18, ϵ is a small positive constant and c is the number of volumetric collisions between the links that are not adjacent in the graph structure or share the same predecessor node.

The summation term is a measure of the pose's ability to explain the point cloud. (Fig. 2.19). The term is based on the logarithmic error of the distance between an observed point, \vec{p}_n , and the nearest surface point of the candidate pose, \vec{p}^* , for all N points in the point cloud and σ is the standard deviation of the points in the point cloud.

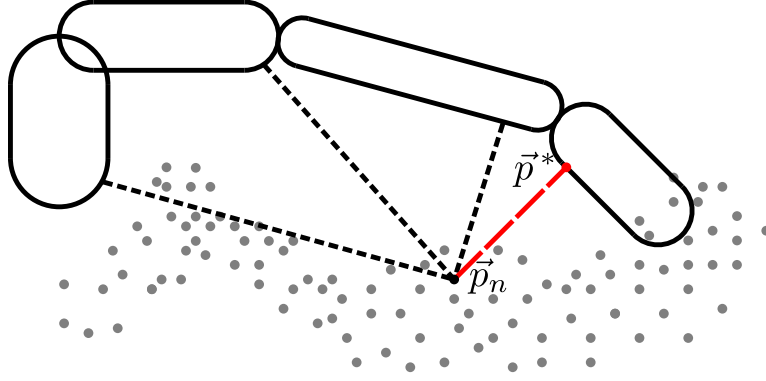


Figure 2.19: A visualization of the fitness metric evaluated for a single point. The distance between the point and the nearest surface is computed for each link, indicated by the dashed lines. Of these distances, the shortest length (highlighted) is used for the fitness calculation (Eq. 2.35).

The nearest surface point of the posed skeleton is defined as the minimum of the nearest surface point for each locally defined link, \vec{p}_j for link j :

$$\vec{p}^*(\theta) = \underset{\vec{p}_j(\theta_j)}{\operatorname{argmin}} \|\vec{p}_n - \vec{p}_j(\theta_j)\| \quad (2.36)$$

The links are composed of a combination of hemisphere and cylinder components and is volumetrically defined using a local representation aligned along the z -axis. The link is the locus of all points satisfying the following conditions:

$$T_0^j \vec{p}_j \Leftarrow \begin{cases} \|\vec{p}_j - \langle 0, 0, r_j \rangle\|^2 = r_j^2 & , \text{if } p_{j,z} < r_j \\ p_{j,x}^2 + p_{j,y}^2 = r_j^2 & , \text{if } r_j \leq p_{j,z} < l_j - r_j \\ \|\vec{p}_j - \langle 0, 0, l_j - r_j \rangle\|^2 = r_j^2 & , \text{if } p_{j,z} > l_j - r_j \end{cases} \quad (2.37)$$

where l_j and r_j are defined in Fig. 2.18.b) and T_0^j is the affine transformation from link j 's frame to the origin.

The fitness metric, Eq 2.35, is related to a maximum likelihood with a heavy-tailed distribution. This distribution was chosen over popular exponential distributions for two reasons. First, the belief distribution from articulated kinematic structures is often multi-modal with isolated peaks. The heavy-tailed distribution allows more inclusive beliefs

while the exponentially bounded distributions are more susceptible to exacerbating the effects of local optima by creating deeper valleys in the fitness landscape.

Second, a kinematic model often does not correspond directly to the visual hull of the depth image subject. Without prior information regarding the subject, the kinematic model only provides a rough approximation of the volumetric subject—details such as mass distribution, deformations at joints and clothing are not captured by kinematic models (compare the synthetic and real data in Fig. 2.23). Exponentially bounded distributions are not sufficiently robust to deal with this gap between the model and reality.

An essential feature of this fitness metric is its data-centric, as opposed to a model-centric, definition. The fitness function is defined strictly by the relationship of the data to the model, and not conversely. The primary benefit of this data-centric definition is its ability to deal with partial self-occlusion in an elegant manner. By avoiding a model-centric likelihood, there is no inherent penalty for positioning occluded links where no data exists. This approach can often lead to the good models by positioning and obstructing individual links such that the remainder of the link chain explains the observed data.

While this fitness metric has numerous advantages from a geometric and modeling perspective, it has many undesirable properties from a machine learning perspective. The fitness metric is not convex and, for articulated subjects, is densely populated with local optima. Furthermore, the parameter space can be extremely large for generalized models. As a result, an evolutionary approach is proposed for this complex machine learning problem.

Genetic algorithms

A genetic algorithm was used to determine the optimal kinematic parameters. Genetic algorithms are stochastic, population-based, heuristic algorithms that iteratively select and recombine solutions to produce increasingly better models. An evolutionary approach provides several benefits to the pose inference problem. First, genetic algorithms have been reliably applied to non-linear, non-convex optimization problems. Next, the population-based dynamics allow for an efficient search of large and high-dimensional parameter spaces. Finally, genetic algorithms are best suited for models with conditionally independent parameters, such as acyclic graphs, as recombination exploits locally optimized subrepresentations.

The population is initialized with randomly generated models: the root node position is initialized on a randomly selected point in the point cloud, the orientation is a quaternion sampled from a Gaussian distribution with a standard deviation of 1 followed by normalization, and the link parameters are interpolating values sampled from a uniform distribution between 0 and 1.

The inference algorithm then progresses via three processes: mutation, recombination and selection. Stochastic point mutations are applied to random parameters in a similar method to the initialization protocol, but localized to individual nodes (Fig. 2.20.a). For recombination, a random crossover point is selected for the existing parent pair, and the offspring are produced by swapping subgraphs (Fig. 2.20.b).

Finally, selection is the process of rejecting inferior models to maintain computational tractability – age-fitness Pareto selection was used [175]. Age-fitness Pareto selection is a selection algorithm that allows for the continuous addition of random individuals to avoid premature convergence. The number of generations that a model

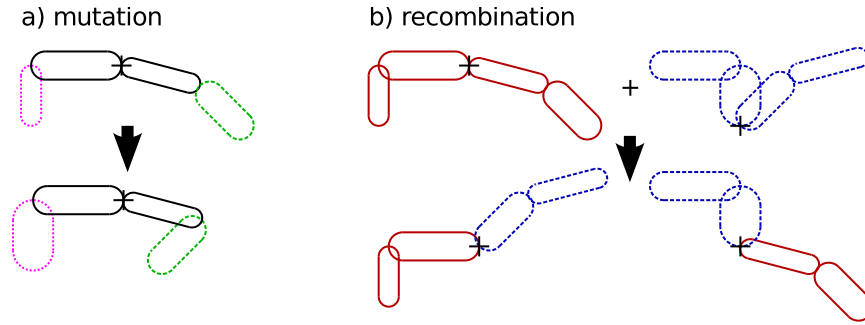


Figure 2.20: A visualization of the a) mutation and b) recombination operators. Mutation changed the parameters of the highlighted links. For recombination, the root was selected as the crossover point and the link chains were swapped to produce offspring.

has existed, or the genotypic age, is logged and a multi-objective Pareto optimization is used to encourage promising individuals while simultaneously also protecting them from being dominated by more mature and optimized solutions. This selection method has been shown to increase the rate of convergence in high-dimensional evolutionary computation domains. For additional details, refer to Section 1.3.4.

Although genetic algorithms are capable of efficiently finding general solutions, convergence to the local optima is often slow. Thus, a stochastic hill-climbing algorithm is applied in each iteration – a random vector is added to the model parameters and the changes are kept only if it results in a higher fitness.

Co-evolutionary rank predictors

A common criticism of evolutionary algorithms, and a prohibitive limitation in practice, stems from the computationally heavy demands of these algorithms. Often, the primary culprit in the computational requirements arises from fitness calculations. In pose inference, determining the fitness of a model requires repeatedly evaluating a local metric. A single point cloud can consist of tens of thousands of points and, since neighbouring

Predictor coevolution example

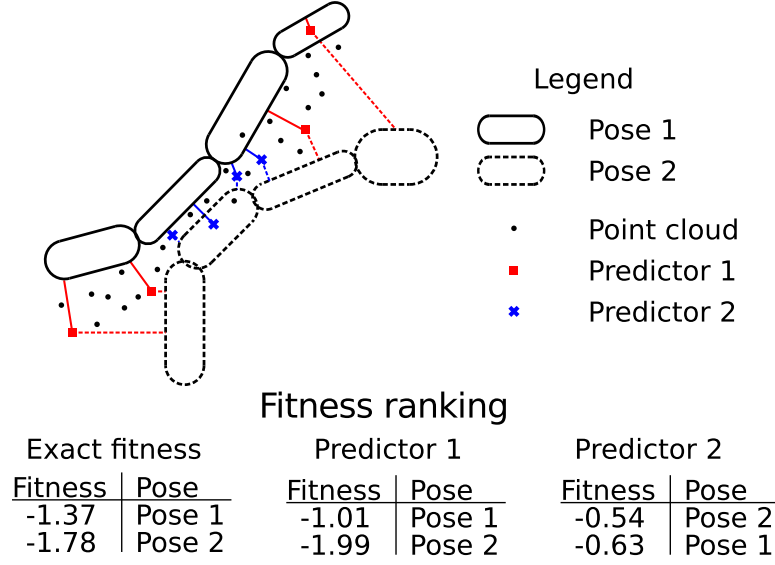


Figure 2.21: An example of predictor co-evolution for pose inference. Two predictors, of four points each, are used to evaluate fitness (Eq. 2.35). Predictor 1 is superior as it obtains the same ranking as the fitness evaluated on the entire data set, while predictor 2 obtains an improper ranking.

points are similar, the computational resources required to calculate the exact fitness results in highly redundant and expensive computations.

Rather than using the entire point cloud, a lightweight approximation is substituted to alleviate the computational demands by co-evolving predictors. In this approach, the fitness is measured only on a dynamic subset of the data, which are co-evolved based on their ability to disambiguate the solution population [25], allowing for evolutionary progress through direct competition. Significant performance acceleration is achieved by a reduction of data in orders of magnitude using this dynamic sampling technique.

For point clouds, predictors are a small subset that references individual points in the point cloud. Fig. 2.21 illustrates a 2D example of predictor co-evolution for point clouds. The original point cloud consists of 32 points and there are two poses with

Algorithm 2.2: Evolutionary pose inference algorithm. Details of rank prediction and age-fitness Pareto selection are found in [175, 176].

```
1 for each model and predictor :
2   initialize with random parameters
3   model.age = 0
4
5 until termination condition :
6   for each randomly selected pair of all models :
7     recombine parents to produce offspring (Fig.2.20b)
8     mutate both offspring (Fig.2.20a)
9     add both offspring to model population
10  for each model :
11    calculate fitness using best current predictor (Eq.2.35)
12    hillclimb each model (Eq.2.35)
13    model.age = model.age + 1
14  insert new random model into population with age = 0
15
16 until model population is reduced to predefined size :
17   for each randomly selected pair of all models :
18     if a model has > age and < fitness than its pair :
19       remove model from population
20
21 for each predictor :
22   for each randomly selected pair of all predictors :
23     recombine parents to produce offspring
24     mutate both offspring
25     calculate fitness = ability to predict model ranking
26     if offspring has >= fitness than parent :
27       replace parent with offspring
28 determine best predictor in population
```

two different predictors. Rather than evaluating the fitness of the poses on the complete point cloud, they are evolved on the predictor subset, which consists of only four points. Simultaneously, the predictors are evolved on their ability to obtain the same fitness ranking as one obtained by using the entire data set—in this example, predictor 1 provides a far superior fitness landscape over predictor 2. This direct competitive co-evolution, along with the reduced computation, greatly increases the solution convergence. For additional implementation details on rank predictor co-evolution, refer to [119, 176].

The complete evolutionary pose inference learning algorithm is summarized in Algorithm 2.2.

2.3.3 Depth image experiments

In this section, the experiments that are used to evaluate the evolutionary pose inference algorithm are described. Both qualitative and quantitative results are shown for two kinematic models and compared against other approaches across a variety of metrics.

Kinematic models

Two distinct models are used to evaluate the learning algorithm. The first is a *spider model*, based on a quadruped robot with 8 links (Fig. 2.17a). The model has 34 degrees of freedom but the links do not overlap workspaces. The second is a *humanoid model*, which consists of 17 links amounting to 78 degrees of freedom (Fig. 2.17b). In addition to the high dimensionality, the links' workspace have significant overlapping regions and there is no constraint on symmetry.

The parameter limits were chosen based on their real-world counterparts, but with an unusually wide range of variability. For example, the humanoid model with mean parameters was based on anatomical body proportions, but can deviate by 25%, which is far beyond the 95th percentile variation in human anatomy [70]. With such a variation in parameter limits, the algorithm is able to represent a wider range of poses than one would expect from real subjects.

Algorithms

As kinematic pose inference this of scope and complexity has been previously considered intractable, there are no related algorithms for direct comparison. Instead the algorithm is compared against baselines along three components:

1. **Sampling method: Co-evolved (C) vs Random (R)** – A comparison of the sampling method used to accelerate the computational performance. Co-evolutionary sampling is the method described in the **Co-evolutionary rank predictors** subsection and is implemented as 8 predictors, each as a subset of 64 points. There were 8 trainers, which were updated every 100 iterations. Random sampling is the baseline that consisted of a single predictor with 64 points selected each generation with a uniform distribution.
2. **Heuristic algorithm: Evolutionary (E) vs Hill-climbing (H)** – A comparison of the heuristic search algorithm. The evolutionary algorithm is the genetic algorithm described in the **Genetic algorithms** subsection. The evolutionary search parameters are: a population of 256 individuals with a mutation and recombination probability of 1% and 50%, respectively. In comparison, there is the hill-climbing alternative which was applied in parallel to 256 initially randomized models.
3. **Kinematic model: Volumetric (V) vs Linear (L)** – A comparison of the kinematic model. The volumetric model is the model described in the **Kinematic models** subsection, while the linear model is a variant that constrained the link radii to zero.

Rather than present every combination of the algorithmic variants, the co-evolved, evolutionary approach with volumetric models is used as a standard and three variants are presented where each component is reduced in a knock-out fashion. A summary of the four approaches; CEV, REV, CHV and CEL; is described in Table 2.3. Furthermore, the initial random population is provided as a baseline (Static) for comparing the effect of inference against random models.

Table 2.3: Pose inference algorithm naming convention

	Sampling		Heuristic		Kinematic	
	Coev.	Rand.	Evol.	Hill	Vol.	Lin.
	(C)	(R)	(E)	(H)	(V)	(L)
1. CEV	×		×		×	
2. REV		×	×		×	
3. CHV	×			×	×	
4. CEL	×		×			×

All inference algorithms began with the same initial, random population. The learning algorithms were executed for 10^9 fitness evaluations, which approximately amounts to 10,000 iterations. On a single core 2.8GHz Intel processor, this required approximately 30 and 70 minutes per image for the spider and humanoid models, respectively.

Synthetic depth data

For a quantitative comparison, a synthetic data set of 128 randomly sampled poses was generated for each model based on the initialization protocol described in the **Genetic algorithms** subsection. A noiseless point cloud was generated via a ray tracing algorithm using 640×480 rays on a field of view of $57^\circ \times 48^\circ$. The model parameters were sampled uniformly, resulting in a varied data set. The baseline algorithms were compared using four metrics:

1. **Fitness metric (Fit.)** The fitness metric which was used for parameter optimization (Eq 2.35).

2. **Mean point distance error (Abs.)** The mean distance between the cloud points and the closest point on the model surface: $E = \frac{1}{N} \sum_{n=0}^N \|\vec{p}^* - \vec{p}_n\|$
3. **Root mean squared point error (RMS)** The root mean squared of the distance between the cloud points and the closest surface point: $E = \sqrt{\frac{1}{N} \sum_{n=0}^N \|\vec{p}^* - \vec{p}_n\|^2}$
4. **Mean joint distance error (Joint)** The inferred joint locations with those from the ground truth model: $E = \frac{1}{J} \sum_{j=0}^J \|\vec{l}_{j,m} - \vec{l}_{j,i}\|$ where $\vec{l}_{j,m}$ and $\vec{l}_{j,i}$ are the j th joint positions for the ground truth model and inferred model, respectively. Note that the cloud data has no direct influence on this metric.

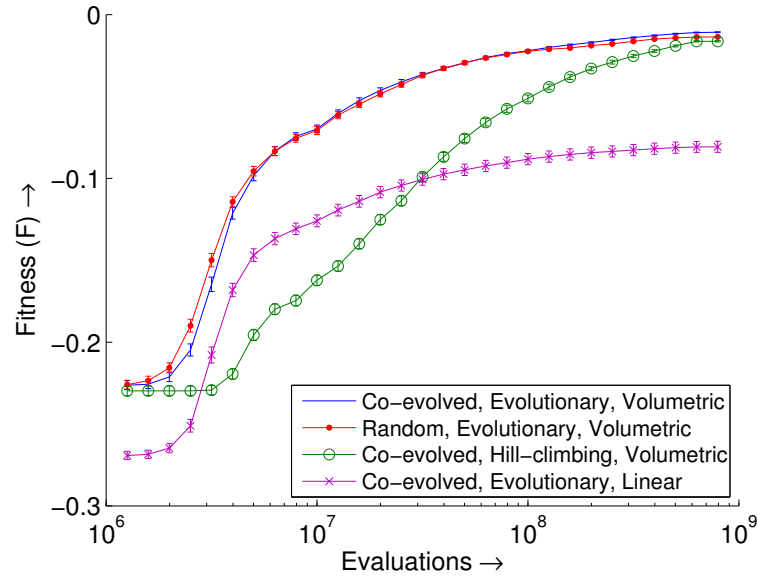
These metrics provide an important basis of comparison as solely relying on the fitness metric presents a skewed perspective. The fitness metric was specifically designed to solve the inference problem. However, due to the logarithmic nature of Eq. 2.35, relative difference in scores are often misleading. Absolute error and RMS provide a more intuitive measure of performance. However, the best metric is the joint error which leverages information from the ground truth to provide an objective measure of performance.

In Table 2.4, the approaches are compared across the various metrics of the best individual after 10^9 fitness evaluations. A single run of each algorithm was performed for each image and the results are averaged over 128 images with standard error reported. By comparing the joint error metric, it is clear that the algorithmic variations are not critical to performance for low-dimensional problems – the problem space is sufficiently small that all four approaches comprehensively cover the search space within the allotted computational effort. Nonetheless, CEV’s performance indicates that it is able to fine tune the parameters in order to achieve the best results over the entire range of metrics.

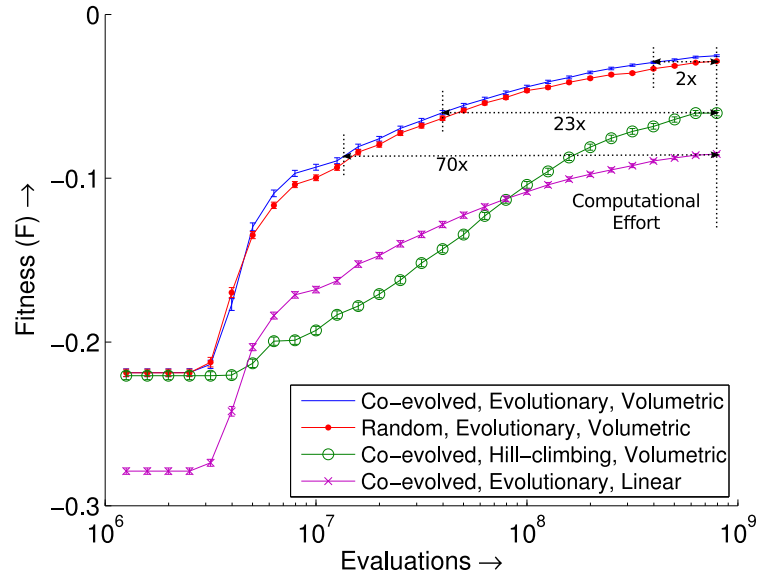
Table 2.4: Pose inference performance on synthetic images

	Spider model			
	Fit.	Abs.	RMS	Joint
	$[\times 10^{-2}]$	$[\times 10^{-3}]$	$[\times 10^{-3}]$	$[\times 10^{-2}]$
CEV	$-1.07 \pm .03$	$1.3 \pm .4$	$2.0 \pm .6$	$8 \pm .7$
REV	$-1.36 \pm .03$	$1.8 \pm .7$	$2.8 \pm .8$	$9 \pm .8$
CHV	$-1.62 \pm .05$	$2.1 \pm .7$	3 ± 1	10 ± 1
CEL	$-8.1 \pm .3$	$8.0 \pm .4$	11 ± 3	10 ± 1
Static	$-22.6 \pm .3$	34 ± 6	53 ± 8	29 ± 2
	Humanoid model			
	Fit.	Abs.	RMS	Joint
	$[\times 10^{-2}]$	$[\times 10^{-2}]$	$[\times 10^{-2}]$	$[\times 10^{-1}]$
CEV	$-2.51 \pm .03$	$1.2 \pm .3$	$1.8 \pm .5$	$1.6 \pm .9$
REV	$-2.85 \pm .06$	$3.1 \pm .8$	$4.8 \pm .7$	$5.6 \pm .8$
CHV	$-6.0 \pm .1$	$3.4 \pm .9$	$5.3 \pm .8$	6 ± 1
CEL	$-8.5 \pm .1$	$4.8 \pm .3$	$6.7 \pm .6$	$8.6 \pm .7$
Static	$-21.8 \pm .3$	56 ± 3	58 ± 6	31 ± 6

However, the algorithmic variations play vital role in inferring the higher dimensional humanoid model. First, CEV is able to achieve the best metric scores, with at least a 3.5-fold improvement in the joint error metric. While REV was able to achieve a similar fitness score to CEV, it is clear that is more susceptible to local optima as it is significantly worse in the other metrics – in fact, REV is only marginally better than CHV. CHV and CEL produced increasingly inferior models, respectively, across the metrics.



(a) Synthetic spider model



(b) Synthetic humanoid model

Figure 2.22: Fitness of the best individual vs. computational effort averaged over 128 images. Error bars indicate standard error ($n = 128$).

In Fig. 2.22, the fitness are compared with respect to the computational effort for both models. Although relative fitness values are misleading and should not be com-

pared directly, fitnesses are still meaningful as benchmarks to measure how fast an algorithm finds equivalently performing models.

For the low-dimensional spider model, CEV and REV achieve similar learning rates, and drastically outperform CHV and CEL. REV is able to provide superior early optimization over CEV, but is overtaken around 10^7 evaluations. This slow start suggests that the competitive co-evolution in CEV requires more overhead to initially build up good individuals in both the solutions and predictor populations, but is able to further optimize the populations when compared to the random subsampling approach.

In the high-dimensional humanoid model, the trends are similar but the discrepancy between the approaches is further amplified. CEV performs significantly better than the other approaches. In fact, for the same final fitness at 10^9 evaluations, CEV provides a 2-, 23- and 70-fold reduction in computation effort over REV, CHV and CEL, respectively.

Note that even with noiseless data generated from identical subjects, the learning algorithm was often unable to find the exact joint positions. However, comparing the poses visually (Fig. 2.23), the inferred models based on synthetic data are very accurate approximations of the original pose. Considering the highly-coupled non-linear model parameters, inferring the ground truth exactly remains a challenge.

Real depth data

The algorithms are compared using depth images captured via a Kinect camera [132]. The Kinect platform was ideal as the consumer hardware is a popular platform for robotic applications with limited accuracy and resolution.

Table 2.5: Pose inference performance on real images

	Spider model		Humanoid model	
	Score [1-5]	LEP [%]	Score [1-5]	LEP [%]
CEV	4.9 \pm .1	1 \pm 1	4.1 \pm .9	16 \pm 4
CHV	4.2 \pm .2	12 \pm 2	3.2 \pm .9	45 \pm 5

For the spider model, a robot with eight limbs with fourteen degrees of freedom was arranged in four distinct poses, and five images ranging in inclination angles were taken per pose. The spider model used the same kinematic structure but had 34 degrees of freedom to account of unknown limb lengths and thickness. The variation in inclination angles provided numerous examples of self-occlusion. For the humanoid model, eight images were taken of four human subjects, totaling to 32 images. The images in both data sets were pre-processed with manual background subtraction so only the pose of interest remained.

For the real depth images, only CEV and REV models were applied as they were the superior approaches from the synthetic data experiments. A single run of each algorithm was performed for each image. Unfortunately, quantitative metrics were unavailable due to the lack of ground truth data. Instead, the resulting models were rated by four volunteers on a scale of 1-5, with 5 as a perfect inference. Furthermore, the number of incorrectly positioned links was reported, and used to calculate the probability of misplacing a link (the Link Error Probability or LEP).

The results are summarized in Table 2.5 with standard error reported. For the low-dimensional spider model, the algorithms are comparable with a slight advantage to CEV in both metrics. However, comparing the high-dimensional humanoid model provides a sharp contrast – REV had difficulties inferring the original pose and often mis-

placed limbs. While REV was able to infer the general layout of the pose, it is more susceptible to misplacing an individual limb.

The difference in the algorithms' performance is evident in the inference examples shown in Fig. 2.23. CEV is able to consistently infer a reasonable approximation to the ground truth, while REV is often caught in spurious local optima that, when rendered, have little in common with the ground truth. The inference algorithm was successful even in cases of significant self-occlusion (Fig. 2.23c,e,g). Although large portions of a limb or the torso were missing, CEV was able to place links in the position of occluded points and infer the correct pose, while REV contorted the kinematic skeleton to find a locally optimal pose.

The inferred models were still reasonable even in CEV's failure modes (an average score lower than 4), especially compared to its REV counterpart. The failure modes were a result of the inference algorithm settling on a local optima within the allotted computational effort. Superior poses might be found with more computational effort, but there is no guarantee of convergence. In these examples, the algorithm had difficulty with the hip orientation, resulting in awkward leg placement. While the failure modes are evident, note that REV fared significantly worse on the same problem.

Finally, additional analysis indicates that predictor co-evolution plays a critical role. By logging which points were referenced, a histogram displaying the frequency that a point was used in the predictor was generated (Fig. 2.24). As the predictor selected 64 points simultaneously, a 375-fold speed-up over using the whole point cloud was obtained in this example. Since points are selected to best disambiguate competing models, a point with higher selection frequency is more useful than its peers for fitness evaluation. The histogram clearly shows a non-uniform distribution, indicating that specific data are more relevant than others.

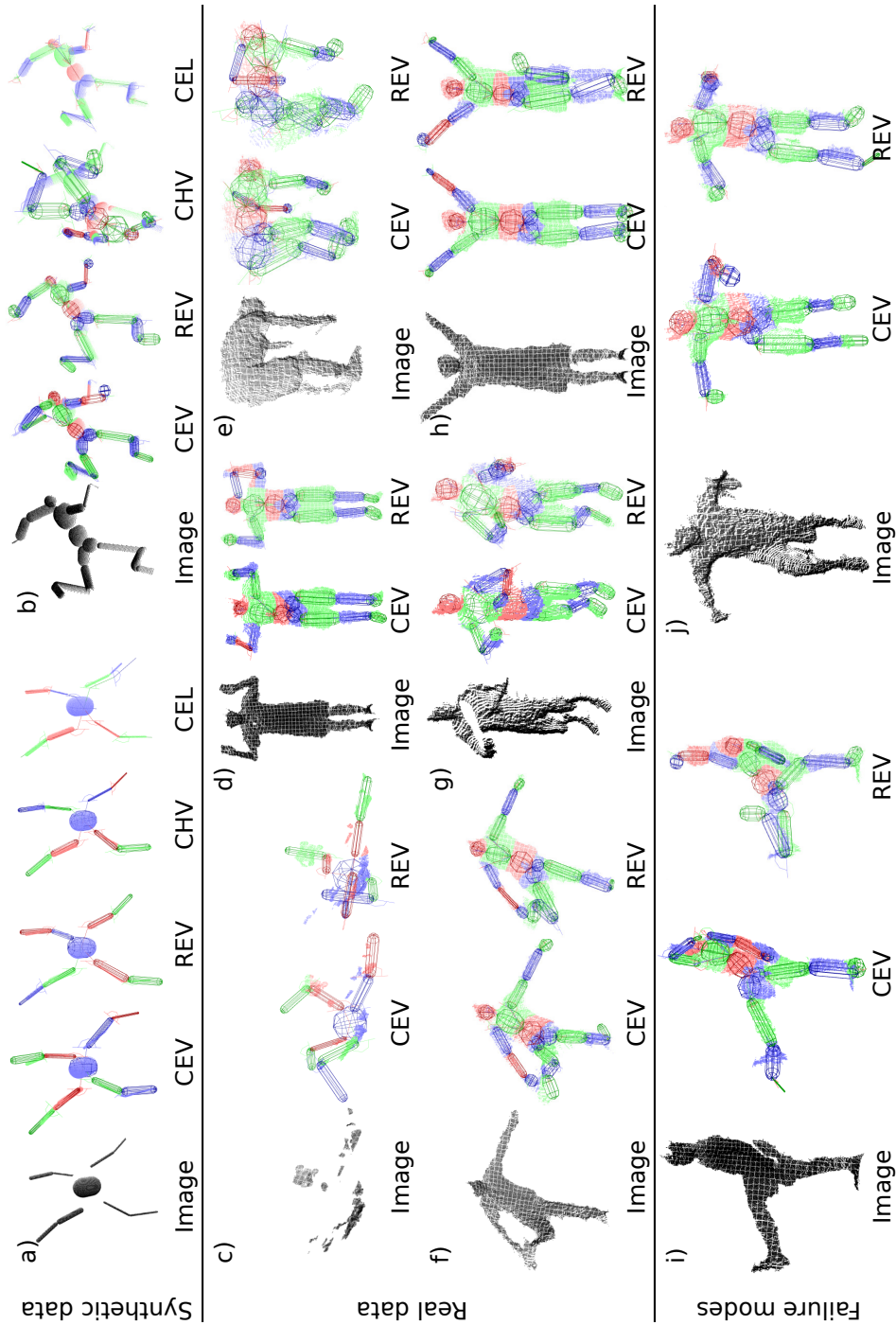


Figure 2.23: Pose inference examples on synthetic and real world data. Note the point clouds are not pre-segmented and the colored links are the result of post-processing for the ease of interpretability.

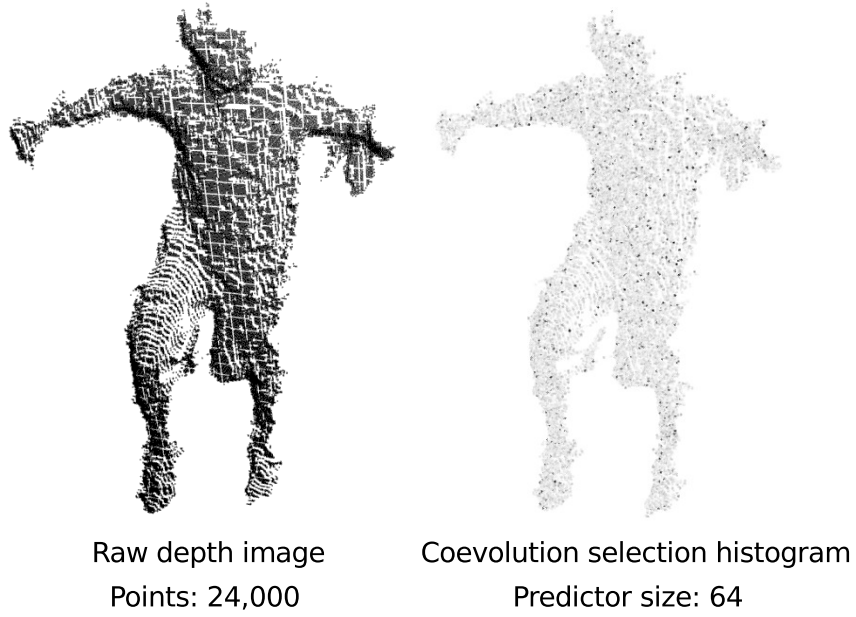


Figure 2.24: A histogram indicating the frequency, proportional to color intensity with darker points being selected more frequently, that a point was selected to be used in a predictor.

2.3.4 Conclusion and future work

The proposed framework of using volumetric kinematic representations and searching for pose parameters using co-evolutionary algorithms based on a heavy-tailed distribution was validated. The poses for 34 degree of freedom spider model and 78 degree of freedom humanoid models were reliably inferred for the synthetic and real RGBD images, even in cases of self-occlusion. The co-evolutionary algorithm achieves a 3.5-fold increase in pose accuracy and a two-fold reduction in computational effort over the baselines. The results indicate that the proposed inference method is vastly superior for high-dimensional problems with articulated links.

Although the algorithm is slower than state of the art methods, it is not dependent on extensive training sets. Rather, this work successfully shows that articulated kinematic structures can indeed be posed in an unsupervised manner, a problem previously

considered intractable. This is a initial step towards fast, unsupervised methods that are more robust than their trained counterparts.

While it is unlikely that kinematic pose inference will be quicker than trained approaches, fast or real-time implementations may be possible. Evolutionary algorithms are naturally parallel and there is room for further optimization. Other areas of interest include using inferred poses to extract kinematic transformations via non-isomorphic structures and tracking poses over a video sequence.

CHAPTER 3

MODELING DISCRETE-CONTINUOUS HYBRID DYNAMICAL SYSTEMS

The problem of creating meaningful models of dynamical systems is a fundamental challenge in all branches of science and engineering. This rudimentary process of formalizing empirical data into parsimonious theorems and principles is essential to knowledge discovery as it provides two integral features: first, the abstraction of knowledge into insightful concepts, and second, the numerical prediction of behavior. While many parametric machine learning techniques, such as neural networks and support vector machines, are numerically accurate, they shed little light on the internal structure of a system or its governing principles. In contrast, symbolic and analytical models, such as those derived from first principles, provide such insight in addition to producing accurate predictions. Therefore, the automated search for symbolic models is an important challenge for machine learning research.

Traditionally, dynamical systems are modeled exclusively as either a continuous evolution, such as differential equations, or as a series of discrete events, such as finite state machines. However, systems of interest are becoming increasingly complex and exhibit a non-trivial interaction of both continuous and discrete elements, which cannot be modeled exclusively in either domain [118]. As a result, hybrid automata, mathematical models which incorporate both continuous and discrete components, have become a popular method of describing a comprehensive range of real-world systems as this modeling technique is perfectly suited for systems that transition between distinct qualitative behaviors. Hybrid dynamical models have been successfully applied to succinctly describe systems in a variety of fields, ranging from the growth of cancerous tumors [4] to air traffic control systems [200].

Although it is plausible to construct hybrid models from inspection and first principles, this process is laborious and requires significant intelligence and insight since each subcomponent is itself a traditional modeling problem. Furthermore, the relationships between every permutation of the subcomponents must be captured, further adding to the challenge. Thus, the ability to automate the modeling of hybrid dynamical systems from time-series data will have a profound affect on the growth and automation of science and engineering.

Despite the variety of approaches for inferring models of time-series data, none are particularly well-suited for building apt descriptions of hybrid dynamical systems. Traditional approaches assume an underlying form and regress model parameters; some approaches conform the data using prior knowledge [56, 149, 206], while others are composed of generalized, parametric, numerical models [9, 30, 103, 109]. Although numeric approaches may be capable of predicting behavior with sufficient accuracy, models of arbitrary systems often require vast numbers of parameters, which obfuscates the interpretability of the inferred model [23]. This trade-off between accuracy and complexity for parametric models is in direct opposition to a fundamental aspect of scientific modeling – abstracting relationships that promote the formulation of new theorems and principles. Thus, constructing symbolic models of hybrid dynamical systems which can be easily and naturally interpreted by scientists and engineers is a key challenge.

The primary contribution of this section is a novel algorithm, called Multi-Modal Symbolic Regression (MMSR), to learn symbolic models of discrete dynamical systems with continuous mappings, as an initial step towards learning hybrid automata. It is a data-driven algorithm that formulates symbolic expressions to describe both the continuous behavior and discrete dynamics of an arbitrary system. Two general learning processes are presented: the first algorithm, Clustered Symbolic Regression (CSR),

generates symbolic models of piecewise functions and the second algorithm, Transition Modeling (TM), searches for symbolic inequalities to model transition conditions. These processes are then applied to a hybrid dynamical systems framework and are used to reliably model a variety of classical problems. MMSR is also applied to infer the modes of operation of a field-effect transistor, similar to those derived from first principles, from measured observations.

The remainder of this section is organized as follows: Section 3.1 provides a brief introduction to hybrid dynamical systems, as well as a description of the relevant work in related fields. Section 3.2 introduces the theoretical background and implementation details of MMSR, with CSR and SR described in Section 3.2.2 and 3.3, respectively. Section 3.4 compares MMSR to traditional machine learning algorithms on four synthetic data sets and presents the inferred transistor model. The section is concluded in Section 3.5.

3.1 Background

This section begins with a brief introduction to the mathematical background of hybrid automata, an inclusive model that describes a variety of hybrid systems. A subset of this general model is described and formulated as the inference target. This is followed by a discussion of the related work in learning hybrid dynamical systems.

3.1.1 Hybrid automata

Due to its inherent complexity, hybrid dynamical systems have only recently emerged as an area of formal research. Consequently, there is a lack of a common

framework, terminology and definition that is universally adopted [21, 77, 205]. This work is based on a popular model called the hybrid automata, which extends the finite automata model to include continuous dynamics. The evolution of the system is deterministic. Each automata, \mathcal{H} , is defined as a 5-tuple, $\mathcal{H} = (\mathcal{W}, \mathcal{X}, \mathcal{M}, \mathcal{F}, \mathcal{T})$, with the following definitions:

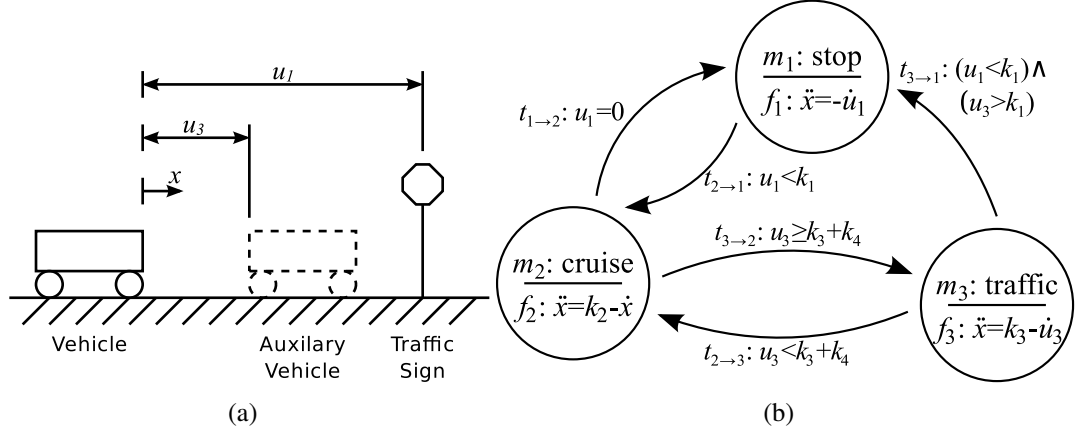


Figure 3.1: An example of a hybrid automata model for a simple, 1D driverless car. A schematic of the system is shown in a) and the system diagram represented as a directed graph is shown in b). \mathcal{W} consists of two inputs, u_1 and u_3 , which corresponds to the distance to the nearest sign and vehicle, respectively; while \mathcal{X} consists of the state variables x and \dot{x} , which describe the vehicle's position and velocity. \mathcal{M} consists of three modes, $\{m_1, m_2, m_3\}$, which represent distinct behaviors corresponding to whether the vehicle is approaching a traffic sign, cruising or driving in traffic; and the behaviors for each mode is described by $\{f_1, f_2, f_3\}$. There are five transitions events, each represented by a Boolean condition.

- \mathcal{W} defines a communication space for which *external variables*, \mathbf{w} , can take their values. The external variables can be further subdivided into *input variables*, $\mathbf{u} \in \mathbb{R}^a$, and *output variables*, $\mathbf{y} \in \mathbb{R}^b$, where $\mathcal{W} = \{\mathbf{u}, \mathbf{y}\}$.
- \mathcal{X} defines a continuous space for which continuous *state variables*, $\mathbf{x} \in \mathbb{R}^c$, can take their values.

- \mathcal{M} defines a countable, discrete set of *modes* in which only a single mode, $m \in \{1, 2, \dots, K\}$, is occupied at a given time. Each mode is represented as a vertex in the system diagram and may have an associated label for ease of reference.
- \mathcal{F} defines a countable, discrete set of first-order, coupled, Differential-Algebraic Equations (DAE). Each equation defines relationship between the state variables, their first-order time derivatives and the inputs:

$$f_k(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{w}) = 0$$

and the solution to these DAE are called the *activities* or *behaviors* of the mode. Each mode, m_k , is defined by its corresponding behavior, f_k , and the solution to the DAE defines the continuous evolution of the system when it is in that mode.

- \mathcal{T} defines a countable, discrete set of *transitions* or *events*, where $t_{k \rightarrow k'}$ denotes a Boolean expression that represents the condition to transfer from mode k to mode k' . If none of the transition conditions are satisfied, then the hybrid automata remains in the current mode and the state variables evolves according to the specified behavior. These are represented as directed edges in the system diagram. For K modes, there are must be at least $K - 1$ transitions and at most K^2 transitions. The transitions defines the discrete evolution of the system by describing how the mode is updated over time.

The challenge in modeling hybrid automata arises from the property that the latent “state” of a hybrid automata depends on both the discrete mode, m_k , as well as the continuous state space vector, \mathbf{x} . As with all dynamical systems, the evolution of the system depends on the initial condition of the latent modes as well as the input variables. An example of a hybrid automata model for a simple, 1D driverless car is illustrated in Fig. 3.1.

3.1.2 Discrete dynamical system with continuous mappings

Hybrid automata are complex models which are capable of describing multi-modal behavior and latent continuous and discrete variables. To restrict the scope of the general problem, a number of assumptions are applied:

1. Each behavior is unique – No two behaviors are the same for any combination of modes: $f_i(\cdot) \neq f_j(\cdot), \forall m_i \neq m_j$.
2. There are no continuous state space variables – All continuous states are directly observable and, thus, the behaviors are defined as strictly input-output relationships, $y = f(u)$, as opposed to DAEs, $y = f(x, \dot{x}, u)$.
3. The number of discrete modes is known – The cardinality of the modes, $|\mathcal{M}| = K$, is provided.

These assumptions describe a continuous-discrete hybrid system that evolves with discrete dynamics but contains continuous input-output relationships. This formulation makes the symbolic inference tractable while also providing a first step to the general solution of inferring hybrid automata. With the exception of assumption 3, each assumption defines a subset of models. If assumption 1 is relaxed, then the model becomes a continuous input-output hidden Markov model [9]. If assumption 2 is further relaxed, then the model becomes the standard hybrid automata described in Section 3.1.1.

The resulting discrete dynamical system with continuous mappings are defined as a 4-tuple, $\mathcal{H} = (\mathcal{W}, \mathcal{M}, \mathcal{F}, \mathcal{T})$, which are time-series models in which the output is dependent on both the observed input as well as the latent mode variable. Furthermore, the evolution of the latent mode variable is dependent on the input via the transition conditions. To continue with the driverless car example, a hybrid automata is transformed into

the desired model by converting the mode and differentiated inputs as explicit inputs and outputs (Fig. 3.2).

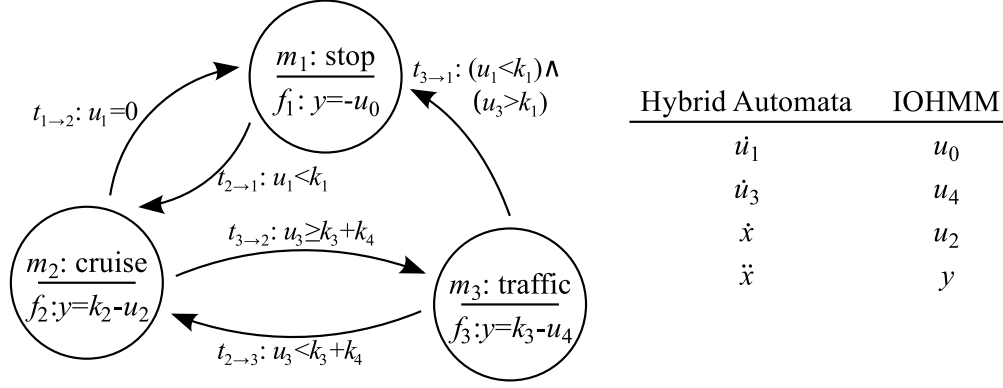


Figure 3.2: A conversion of the 1D driverless car hybrid automata as a discrete dynamical model with continuous mappings. The system diagram and variable conversion are shown.

3.1.3 Related work

Although there is little work in the automated, data-driven construction of models of hybrid dynamical systems whose components are expressed in symbolic mathematics, there are many machine learning approaches that are capable of describing and predicting the behavior of multi-modal, time-series data via alternate approaches.

Interest in hybrid dynamical systems is primarily spurred by the control systems community and consequently, they have proposed a variety of approaches to infer dynamical systems. One approach addresses the problem of modeling discrete-time hybrid dynamical systems by reducing the problem to switched, piecewise affine models [56, 206], and procedures using algebraic, clustering-based, Bayesian, and bounded-error methods have been proposed [149]. This modeling technique imposes a linear form to the system's dynamics, which substantially simplifies the modeling but limits

the explanatory range of such models by enforcing linear approximations to non-linear systems.

Another approach uses non-linear, generalized, parametric models to represent hybrid dynamical systems. Chen et al. modeled hybrid systems using radial basis function networks [30], while Le et al. used support vector machines to approximate non-linear systems [109]. The primary limitation with this approach is its reliance on parametric modelling in the form of neural networks or support vector machines. While parametric models can produce arbitrarily accurate predictions, they often require a vast quantity of weight parameters to achieve such accuracies in non-trivial systems. This tradeoff between accuracy and complexity often results in uninterpretable models which makes it difficult to extract meaningful relationships from the data [23].

Recurrent Neural Networks (RNNs) have been a popular approach for modeling time-series data. The input-output relationship of a general, continuous dynamical system has been modeled with RNNs [103] and it has also been shown that recurrent neural networks are capable of modeling finite state machines [81]. However, there has been no reported work on specifically modeling discrete dynamical systems with continuous inputs and outputs. RNNs are also restricted by their parametric nature, often resulting in dense and uninterpretable models.

Consequently, there has been significant interest in extracting rules from parametric, recurrent neural networks to build a formal, symbolic model and providing an important layer of knowledge abstraction [143, 144, 145, 202]. However, a recent review of this work suggests that there is limited progress in handling RNNs with non-trivial complexity [87].

The learning of input-output hidden Markov machines has been previously studied by Bengio and Frasconi [9], which uses architecture based on neural networks to predict both the output of each mode as well as the transition conditions. The generalized expectation-maximization algorithm is used to optimize the parameters of in each of neural networks. Although the original work was implemented on grammatical inference with discrete inputs and outputs, the framework has since been adapted to several applications in the continuous domain [69, 125]. However, these approaches also rely on complex parametric neural network models.

This technique attempts to resolve these various challenges by building models of hybrid dynamical systems that uses non-linear symbolic expressions for both the behaviors as well as the transitions. Rather than imposing a linear structure or using parametric models, symbolic expressions are inferred to provide a description that is in the natural, mathematical representation used by scientists and engineers.

3.2 Symbolic regression of piecewise functions

This section begins with a formalization of the learning problem. This is followed by the description of two, general algorithms: clustered symbolic regression and transition modelling. The section is concluded by combining both subalgorithms within a hybrid dynamics framework to form the multi-modal symbolic regression algorithm.

3.2.1 Problem formalization

The goal of the algorithm is to infer a symbolic discrete dynamics model with continuous mappings from time-series data, where symbolic mathematical expressions are

learned for both the behaviors as well as the transition events. Consider a dynamical system that is described by:

$$m_n = T(m_{n-1}, \mathbf{u}_n),$$

$$\mathbf{y}_n = F(m_n, \mathbf{u}_n) = \begin{cases} F_1(\mathbf{u}_n) & , \text{ if } m_n = 1 \\ \vdots & , \quad \quad \quad \vdots \\ F_K(\mathbf{u}_n) & , \text{ if } m_n = K \end{cases}$$

where $\mathbf{u}_n \in \mathbb{R}^p$ is the input vector at time n , $\mathbf{y}_n \in \mathbb{R}^r$ is the output vector, and $m_n \in M = \{1, 2, \dots, K\}$ is the mode state.

The goal is to infer a multi-modal, input-output model that minimizes the normalized, negative log probability of generating the desired output vector under a mixture of Laplacians model, E , over the time series:

$$E = \frac{1}{N} \sum_{n=1}^N -\ln \left(\sum_{k=1}^K \gamma_{k,n} e^{-\frac{\|\mathbf{y}_n - \hat{\mathbf{y}}_{k,n}\|}{\sigma_y}} \right) \quad (3.1)$$

where $\gamma_{k,n} = p(m_n = k)$ or the probability that the system is in mode k , $\hat{\mathbf{y}}_{k,n}$ is the output of function $F_k(\mathbf{u}_n)$, and σ_y is the standard deviation of the output data.

This error metric is adapted from related work by Jacobs et al. on mixtures of local experts [86], but with the assumption of Laplacian, as opposed to Gaussian, distributions. The Laplacian distribution was chosen due to its relationship to absolute error, rather than squared error for Gaussian distributions. Note that for true mode probabilities or uni-modal models, this error metric indeed reduces to normalized, mean absolute error. Mean absolute error was preferred over squared error as is it more robust to outlier errors that occur due to misclassification.

To learn symbolic models of discrete dynamical systems with continuous mappings, the Multi-Modal Symbolic Regression (MMSR) algorithm is composed of two general algorithms: Clustered Symbolic Regression (CSR) and Transition Modelling (TM).

CSR is used to cluster the data into symbolic subfunctions, providing a method to determine the modal data membership while also inferring meaningful expressions for each subfunction. After CSR determines the modal membership, TM is then applied to find symbolic expressions for the transition conditions.

This algorithm varies from traditional learning approaches for hidden Markov model; conventional Baum-Welch or forward-backward algorithms are insufficient for dealing with the input-output relationships and transition conditions. Bengio and Frasconi approached the learning challenge by introducing the Generalized Expectation-Maximization (GEM) to find the optimum parameters for the input-output functions and transition conditions simultaneously [9]. However, for non-trivial, continuous systems, the GEM approach is likely to settle on local optima due to the inability of transition modelling to discriminate distinct modes. By dividing the problem into the CSR and TM subdomains, our approach leverages the property that each behavior is unique to infer accurate and consistent hybrid dynamical systems.

3.2.2 Clustered symbolic regression

The first algorithm is Clustered Symbolic Regression (CSR), which involves using unsupervised learning techniques to solve the challenging issue of distinguishing individual functions while simultaneously infers a symbolic model for each of them. This novel algorithm is presented as a generalized solution to learning piecewise functions, distinct from the hybrid dynamics framework.

This subsection begins with a formal definition of the problem, followed by a brief overview of two learning approaches: Symbolic Regression (SR) and Expectation-

Maximization (EM), respectively. These approaches are then unified as clustered symbolic regression.

Problem definition

Consider the following, generalized piecewise function:

$$\mathbf{y}_n = f(\mathbf{u}_n) = \begin{cases} f_1(\mathbf{u}_n) & , \text{ if } \mathbf{d}_n \in D_1 \\ \vdots & , \quad \quad \quad \vdots \\ f_K(\mathbf{u}_n) & , \text{ if } \mathbf{d}_n \in D_K \end{cases}$$

where $\mathbf{u}_n \in \mathbb{R}^p$ is the observable input vector at index n , $\mathbf{y}_n \in \mathbb{R}^r$ is the output vector, $\mathbf{d}_n \in \mathbb{R}^q$ is the domain input vector and D is a set of mutually exclusive membership subdomains. The domain input vector can be composed of both the observable variables, \mathbf{u}_n , as well as latent variables, allowing for latent subdomains definitions. Given the number of subdomains, K , infer a model that minimizes the within-domain, absolute error, E_{CSR} :

$$E_{CSR} = \sum_{n=1}^N \sum_{k=1}^K \gamma_{k,n} \|\mathbf{y}_n - \hat{\mathbf{y}}_n\| \quad (3.2)$$

where $\gamma_{k,n}$ is the probability that the input-output pair belongs to the subdomain D_k and $\hat{\mathbf{y}}_n$ are model predictions of the output at time n .

In essence, this formulation is an unsupervised clustering problem. However, unlike traditional clustering problems, each cluster is represented by a symbolic expression and there is no prior knowledge regarding the structure of these submodels. There has been no reported work on mixture models where each component model is dependent on an arbitrary functional of the input; conventional mixture models assume that each cluster belongs to the same fixed-structure, parametric family of distributions [11].

Symbolic regression

The first component of CSR is Symbolic Regression (SR): an genetic programming algorithm that is capable of reliably inferring symbolic expressions that models a given data set [38, 46, 104, 142]. Provided with a collection of building blocks and a fitness metric, SR attempts to find the combination of primitives that best maximizes the stated fitness function. For additional information regarding symbolic regression, refer to Section 1.3.2.

SR was chosen as the modeling algorithm because it provides three unique advantages. First, SR includes form and structure as part of the inference problem. Free form expressions are generated by rearranging primitives in a boundless tree structure, resulting in a rich range of possible expressions. In contrast, parametric models constrict their solution space to sums of basis and transfer functions.

Next, SR produces solutions that are easily interpreted. Unlike other machine learning algorithms which tweak a vast collection of intangible numerical parameters, symbolic expressions are the foundation of mathematical notation and often provide key insight into the fundamental relationships of such models.

Finally, an important quality of SR is its natural method to deal with overfitting, where the inferred model captures the peculiarities of the data set rather than the underlying truth. Overfitting is a major issue in machine learning and, to an even greater extent, multi-modal systems where one overfit behavior can cripple the progress in the remaining behavior. In symbolic expressions, overfitting occurs by inferring a model with greater complexity than the ground truth – for example, a cubic model is used to fit a quadratic function. Thus, there is a fundamental trade-off between the accuracy

and complexity of a candidate model, where overfitting incurs additional complexity to accurately model the noise contributions in the data.

Instead of simply reporting the most accurate model found by SR, which is susceptible to overfitting, the inherent population dynamics is leveraged to provide a multi-objective approach to dealing with overfitting. By design, SR generates numerous candidate models with varying degrees of complexity and accuracy. Rather than considering every generated expression as a candidate model, individual expressions are compared against a continuously updated, multi-objective record. This approach, called Pareto optimization, forms a set of non-dominated solutions which provide the best fitness for a given complexity. This method reformulates the problem of overfitting as model selection along the accuracy and complexity trade-off, a property later exploited by CSR to reliably find solutions. As Pareto optimization is a post-processing technique that analyzes expressions only after they have been generated by SR, it does not interfere with the underlying search process. For additional information regarding symbolic regression, refer to Section 1.3.4.

Recent advances in SR implementations have made it a powerful search tool – even for difficult search spaces, it can often find good, if not globally optimal, solutions. It is capable of non-linear regression and has even been shown to find differential equations [85], implicit equations [174] and even conservation laws [173]. Despite the range of successful applications, SR is limited to individual functions. Currently, there are no algorithms that are capable of symbolically regressing unlabeled data generated by multi-modal systems, such as data from a hybrid dynamical system or piecewise function.

Algorithm 3.1: Generalized expectation-maximization

```
1 input → observed data -  $X$ 
2 output → model parameters -  $\theta$ 
3
4 initialize model parameters -  $\theta_{\text{old}}$ 
5 while convergence is not achieved :
6     # expectation step
7     compute probability of observing latent variables -  $p(Z|X, \theta_{\text{old}})$ 
8     # maximization step
9     compute new model parameters -  $\theta_{\text{new}} = \arg \max_{\theta} \sum_Z p(Z|X, \theta_{\text{old}}) \ln p(X|Z, \theta)$ 
10    update model parameters -  $\theta_{\text{old}} = \theta_{\text{new}}$ 
11 return model parameters  $\theta_{\text{old}}$ 
```

Expectation-maximization

The second component of CSR is the Expectation-Maximization (EM) algorithm: a machine learning algorithm that searches for the maximum likelihood estimates of model parameters and latent variables. Formally, the EM solves the following problem: given a joint distribution $p(X, Z|\theta)$ over observed variables X and latent variables Z , governed by the model parameters θ , determine the parameter values that maximize the likelihood function $p(X|\theta)$ [45].

The EM algorithm is an iterative two-step process, which begins with initially random model parameters. In the expectation step, the expected value of the latent variables is determined by calculating the log-likelihood function given the current model parameters. This is followed by the maximization step, where the model parameters are chosen in order to maximize the expected value given the latent variables. Each cycle of EM increases the incomplete-data log-likelihood, unless it is already at a local optimum. The implementation details of EM are summarized in Algorithm 3.1.

The EM algorithm is a popular framework for a variety of mixture models, including mixture of Gaussians, mixture of Bernoulli distributions and even Bayesian linear regression [11]. Although evolutionary computation has been applied to the EM frame-

work [126, 152], the focus has been on exploring different optimization approaches in the maximization step for mixture of Gaussians, as opposed to investigating different types of models found through evolutionary computation.

3.2.3 Clustered symbolic regression algorithm

The Clustered Symbolic Regression (CSR) is a novel algorithm that is capable of finding symbolic expressions for piecewise functions. By applying an EM framework to SR, this algorithm determines both the model parameters, mathematical expressions and the corresponding variances for each subfunction, as well as the latent variables, the membership of each data point, for a piecewise function.

To aid in the formulation of the algorithm, the SR optimization is interpreted in a statistical framework where the output of each subfunction defines the expected value conditional on the input and state, $f_k(\mathbf{u}_n) = E[\mathbf{y}_n | m_k, \mathbf{u}_n]$, where m_k is 1 if $\mathbf{d}_k \in D_k$ and 0 otherwise. Assuming that the noise follows a Gaussian distribution, then the following definition is obtained:

$$p_k(\mathbf{y}_n | \mathbf{u}_n) = \mathcal{N}(\mathbf{y}_n | f_k(\mathbf{u}_n), \sigma_k^2) \quad (3.3)$$

where $\mathcal{N}(x | \mu, \sigma^2)$ defines a Gaussian distribution over x with a mean μ and variance σ^2 .

The expectation step consists of evaluating the expected membership values using the current model. Using the probabilistic framework for defining functions (Eq. 3.3), the probability of membership, $\gamma_{k,n}$, of an input-output pair to a function f_k is:

$$\gamma_{k,n} = \frac{\mathcal{N}(\mathbf{y}_n | f_k(\mathbf{u}_n), \sigma_k^2)}{\sum_{k=1}^K \mathcal{N}(\mathbf{y}_n | f_k(\mathbf{u}_n), \sigma_k^2)}. \quad (3.4)$$

Note that the membership probability reinforces exclusivity – given two subfunctions with the same expression, the model with lower variance has stronger membership values over the same data. This property is advantageous given the assumption that each behavior is unique; as one subfunction becomes increasingly certain as a result of a decreased variance, the other subfunctions are forced to model the remaining data.

Next, the Maximization Step consists of finding the expressions for each behavior and variances that best explain the data points given the current membership distribution. The variance of each behavior is updated by computing the unbiased, weighted sample variance using the functions obtained by SR (Eq. 3.5).

$$\sigma_k^2 = \frac{\sum_{n=1}^N \gamma_{k,n}}{(\sum_{n=1}^N \gamma_{k,n})^2 - \sum_{n=1}^N \gamma_{k,n}^2} \sum_{n=1}^N \gamma_{k,n} \|y_n - f_k(\mathbf{u}_n)\|^2 \quad (3.5)$$

To find the behavior for each mode, SR is used to efficiently find the most suitable expression for the subfunction relationship:

$$y_n = f_k(\mathbf{u}_n). \quad (3.6)$$

Although the CSR is designed to optimize the weighted absolute error (Eq. 3.2), that fitness metric is not ideal for each local SR search. The individual data sets, described by membership probabilities, contain both measurement noise as well as classification error. Data from erroneous classifications often produces heavy-tail outliers. Thus, each local search requires a robust metric that does not assume exponentially bound likelihoods – the weighted, mean logarithmic error was selected as the fitness metric:

$$F_{\text{local},k} = -\frac{\sum_{n=1}^N \gamma_{k,n} \log(1 + \|y_n - f_k(\mathbf{u}_n)\|)}{\sum_{n=1}^N \gamma_{k,n}}. \quad (3.7)$$

However, applying the logarithmic fitness naively tended to bias the SR search to find the same expression for every behavior in the initial iteration, resulting in a symmetrical, local optimum for the EM algorithm. A greedy implementation of the EM

algorithm that updates each behavior sequentially was used to resolve this issue. This approach enforces a natural priority to the learning algorithm allowing each behavior to model the data of its choice, forcing the remaining functions to model the remaining data.

A primary issue with the EM algorithm is that it is sensitive to initial conditions and likely to settle on local optima. Local optima occurs when some solutions are overfit, which results in underfit solutions for the remaining behaviors. By exploiting the set of solutions provided by Pareto optimization, CSR is significantly more robust to initial conditions and able to find the global optima with greater consistency. Assuming the Pareto optimal set is exhaustive, if an overfit solution exists in the Pareto optimal set, then the true and less complex solution also exists in the set. Thus, the challenge of avoiding local optima due to overfitting is reduced to selecting the most appropriate solution from this set.

Each solution in the Pareto optimal set is selected, temporary membership, $\gamma'_{k,n}$, and variances, $\sigma_k'^2$, are calculated and the global error is determined (Eq. 3.2). The global error is used to compute the Akaike Information Criterion score [2], a metric that rewards models that best explain the data with the least number of parameters:

$$AIC = 2c + N \log |E_{CSR}| \quad (3.8)$$

where c is the number of nodes in the tree expression and N is the number of data points. The solution with the lowest global AIC score is deemed to have the most information content, and thus, is the most appropriate solution. Although other information based methods are available [187, 210], AIC was used because of its ease of application.

Note that while the AIC score is used for model selection, using this metric directly in the SR as a fitness function often leads to inferior results as it biases the search space to look for simple solutions which can lead to underfit models. Instead, this approach

Algorithm 3.2: Clustered symbolic regression

```
1  input  → unclustered input-output data -  $\mathbf{u}_n, \mathbf{y}_n$ 
2          → the number of subfunctions -  $K$ 
3  output → behavior for each mode -  $f_k(\mathbf{u}_n)$ 
4          → variance for each mode -  $\sigma_k^2$ 
5
6  function symbolic_regression(search_relationship, fitness_function) :
7      initialize population with random expressions defined by search_relationship
8      for predefined computational effort :
9          generate new expressions from existing population (Fig.1.2)
10         calculate fitness of all expressions according to fitness_function
11         remove unsuitable expressions from the population
12         for each pop_expr in the population :
13             for each pareto_expr in the pareto_set :
14                 if ((pop_expr.fitness > pareto_expr.fitness) and
15                     (pop_expr.complexity <= pareto_expr.complexity)) :
16                     add pop_expression to pareto_set
17                     remove pareto_expression from pareto set
18     return pareto_set
19
20 initialize random membership values
21 for each behavior in  $K$  modes :
22     sr_solutions = symbolic_regression(Eq.3.6, Eq.3.7)
23     set behavior  $f_k$  to solution with lowest local AIC score in sr_solutions
24 set variance for each behavior -  $\sigma_k^2$  (Eq.3.5)
25 while convergence is not achieved :
26     for each behavior in  $K$  modes :
27         # expectation step
28         for all the  $N$  data points :
29             compute membership values -  $\gamma_{k,n}$  (Eq.3.4)
30         # maximization step
31         sr_solutions = symbolic_regression(Eq.3.6, Eq.3.7)
32         for each solution in sr_solutions :
33             compute temporary membership values -  $\check{\gamma}_{k,n}$  (Eq.3.4)
34             compute temporary variance -  $\check{\sigma}_k^2$  (Eq.3.5)
35             compute global fitness using temporary values -  $E_{CSR}$  (Eq.3.2)
36             compute AIC score using global fitness (Eq.3.8)
37             set behavior  $f_k$  to solution with lowest AIC score in sr_solutions
38             set variance to corresponding value -  $\sigma_k^2$  (Eq.3.5)
39 return behaviors  $f_k$  and variances  $\sigma_k^2$ 
```

of focusing solely on model accuracy, populating a set of candidate solutions ranging in complexity and then using the AIC to select from this list proved to produce the most consistent and reliable models.

The complete CSR algorithm is summarized in Algorithm 3.2.

3.3 Modeling transition conditions

The second algorithm is Transition Modeling (TM), which is a supervised learning technique that determines symbolic, discriminant inequalities for transition events by restating a classification problem as a regression problem using function composition. This algorithm is presented as a generalized solution to classification with symbolic expressions, separate from the hybrid dynamics framework. This subsection begins with a formal definition of the problem, followed by a discussion of related work and description of the algorithm.

3.3.1 Problem definition

Consider the general, binary classification problem:

$$\zeta_n = \begin{cases} 1 & , \text{ if } \mathbf{u}_n \in Z \\ 0 & , \text{ if } \mathbf{u}_n \notin Z \end{cases} \quad (3.9)$$

where $\mathbf{u}_n \in \mathbb{R}^p$ is the input vector at index n , $\zeta_n \in \mathbb{B}$ is the corresponding label and Z is the characteristic domain. Infer the discriminant function which describes the characteristic domain that minimizes the classification error:

$$E_{TM} = \sum_{n=1}^N \|\zeta_n - \hat{\zeta}_n\|. \quad (3.10)$$

Despite the variety of approaches to binary classification, the explicit requirements for a non-linear, symbolic model of the discriminant function makes this problem challenging. While multiclass classification may be more appropriate, it results in a significantly more challenging problem for symbolic models and is left for future work.

3.3.2 Related work

Although using evolutionary computation for classification has been previously investigated, this algorithm is novel due to its reformulation of the classification problem as symbolic regression, providing an assortment of benefits.

The majority of classifying evolutionary algorithms impose a fuzzy logic structure with triangular or trapezoidal membership domains [6, 88, 130]. A genetic algorithm is then used to optimize the parameters of these fixed-structure discriminant functions. This technique is difficult to scale to non-linear, multi-inputs domains as it only searches for the model parameters using a fixed model structure. Furthermore, the solutions may be difficult to interpret or express succinctly as the number of domains increases.

Muni et al. designed an evolutionary program that is capable of generating symbolic expressions for discriminant functions [136]. This program was limited to a classification framework, resulting in application-specific algorithms, fitness metrics and implementations. This approach is novel as it adapts the well-developed framework of SR, allowing for a unified approach to both domains.

3.3.3 Transition modeling algorithm

The Transition Modeling (TM) algorithm builds on the infrastructure of SR. The discriminant functions are expressed symbolically as an inequality, where the data has membership if the inequality evaluates to true. For example, the inequality $Z(u) : u \geq 0$ denotes the membership for positive values of u , while $Z(u_1, u_2) : u_1^2 + u_2^2 \leq r^2$ describes membership for an inclusive circle of radius r . The key insight in reforming the classification problem into a regression problem is that function composition with a Heavyside

step function is equivalent to searching for inequalities:

$$\zeta = \text{step}(x) = \begin{cases} 1 & , x \geq 0 \\ 0 & , x < 0 \end{cases}.$$

Using the step function and function composition, the classification problem (Eq. 3.9) is reformatted as a standard symbolic regression problem using the search relationship:

$$\zeta_n = \text{step}(Z(\mathbf{u}_n)).$$

This reformulation allows a symbolic regression framework to find for symbolic, classification expressions, $Z(\cdot)$, that define membership domains. The expression is readily transformed into an inequality, $Z(\cdot) \geq 0$, allowing for natural interpretation.

Although the step function illustrates the relationship between TM and SR, it is actually difficult to use in practice due to the lack of gradient in the fitness landscape. Small perturbations in the expression are likely to have no effect on the fitness, which removes any meaningful incremental contributions from gradient dependent techniques, such as hill climbing. Thus, searching with step functions requires that the exact expression is found through the stochastic processes of recombination and mutation, which may lead to inconsistent results and inefficient computational effort. Instead, a function composition with the sigmoid (Eq. 3.11) was found to be more practical as a ‘soft’ version of the step function, leading to the search expression in Eq. 3.12 while still using the fitness metric (Eq. 3.10).

$$\text{sig}(x) = \frac{1}{1 + e^{-x}} \tag{3.11}$$

$$\zeta_n = \text{sig}(Z(\mathbf{u}_n)) \tag{3.12}$$

The sigmoid function provides three important benefits. First, it provides a quantified measure of degree of belief. In the limit of $|x| \rightarrow \infty$, the sigmoid function approaches the step function. Thus, the magnitude of the scaling factor in $Z(\cdot)$ provides a numerical measure of the certainty of the classifier; confident classifiers have expressions with large scaling factors. Furthermore, for ease of interpretability, the scaling factor is easily removed via algebraic simplifications. The second benefit is that sigmoid TM provides an elegant method to deal with uncertain or fuzzy memberships. Since the sigmoid is a continuous function ranging from 0 to 1, it is able to represent all degrees of membership as opposed to purely Boolean classification. The final benefit is inherited from SR: a range of solutions is provided via Pareto optimality, balancing model complexity and model accuracy, and model selection is used to prevent overfit solutions.

3.3.4 Modeling hybrid dynamical systems

To infer symbolic models of hybrid dynamical systems, two general CSR and TM algorithms are applied to form the Multi-Modal Symbolic Regression algorithm (MMSR). CSR is first used to cluster the data into distinct modes while simultaneously inferring symbolic expressions for each subfunction. Using the modal membership from CSR, TM is subsequently applied to find symbolic expressions for the transition conditions. Of the 4-tuple description of in Section 3.1.2, $\mathcal{H} = (\mathcal{W}, \mathcal{M}, \mathcal{F}, \mathcal{T})$, the communication space, \mathcal{W} , is provided by the time-series data and it is the goal of MMSR to determine the modes, \mathcal{M} , behaviors, \mathcal{F} , and transitions, \mathcal{T} .

Using the unlabeled time-series data, the first step is to apply CSR. CSR determines the modes of the hybrid system, \mathcal{M} , by calculating the membership of an input-output

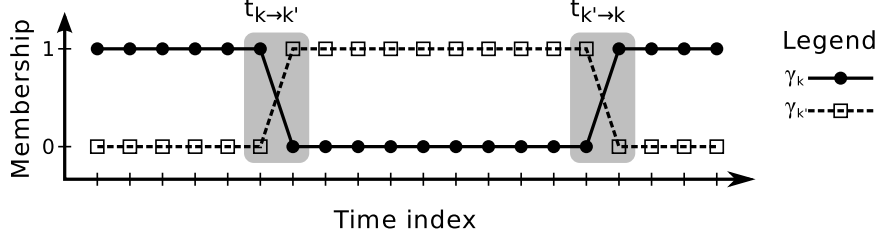


Figure 3.3: An example of the time-series data of membership signals. Transitions are highlighted in grey.

pair (expectation step of Algorithm 3.2). Simultaneously, CSR also infers a non-linear, symbolic expression for each of the behaviors, \mathcal{F} , through weighted symbolic regression (maximization step of Algorithm 3.2).

Using the modal memberships from CSR, TM searches for symbolic expressions of the transition events, \mathcal{T} . To find the transitions, the data must be appropriately pre-processed within the hybrid system framework. Transition events are defined as the conditions for which the system moves from one mode to another. Using the membership values from CSR to determine the mode at every data point, searching for transition events is rephrased as a classification problem: a transition from mode k to mode k' occurs at index n if and only if $\gamma_{k,n} = 1$ and $\gamma_{k',n+1} = 1$ (Fig. 3.3). Thus, the classification problem is applied to membership levels of the origin and destination modes. For finding all transition events from mode k to mode k' , the search relationship and fitness metric are respectively:

$$\gamma_{k',n+1} = \text{sig}(t_{k \rightarrow k'}(\mathbf{u}_n)),$$

$$F_{\text{transition}} = - \sum_{n=1}^{N-1} \gamma_{k,n} \|\gamma_{k',n+1} - \text{sig}(t_{k \rightarrow k'}(\mathbf{u}_n))\|^2$$

It is important to realize that most data sets are heavily biased against observing transitions – the frequency at which a transition event occurs, or a Positive Transition Point (PTP), is relatively rare compared to the frequency of staying in the same node, or

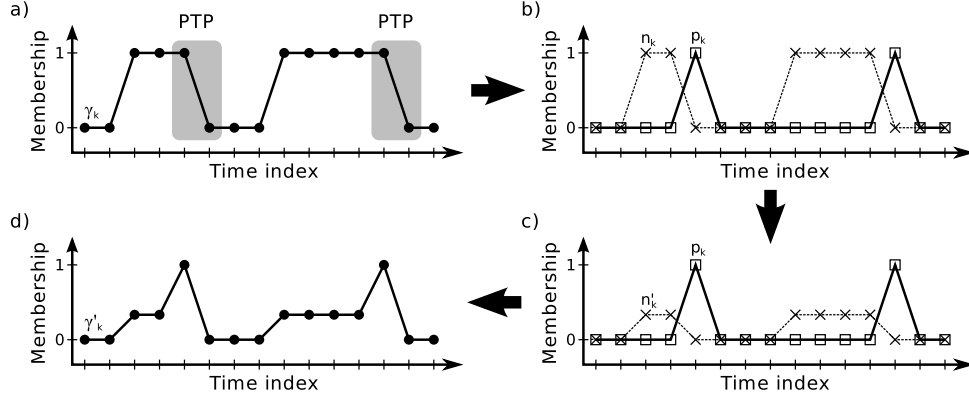


Figure 3.4: An example of PTP-NTP weight balance. a) Original weight data ($\gamma_{k,n}$). b) Weight data decomposed into $p_{k,n}$ and $n_{k,n}$ signals. c) Scaled $\tilde{n}_{k,n}$ signal. d) $p_{k,n}$ and $\tilde{n}_{k,n}$ recombined to form balanced ($\tilde{\gamma}_{k,n}$).

Negative Transition Point (NTP). A PTP is defined mathematically for mode k at index n if $\gamma_{k,n} = 1$ and $\gamma_{k,n+1} = 0$; all other binary combinations of values are considered NTPs. This definition is advantageous since PTPs are identified by only using the membership information of only the current mode, $\gamma_{k,n}$, and no other membership information from the other modes are required.

The relative frequencies of PTP and NTP affects the TM algorithm since the data set is imbalanced: the sum of the weights associated with NTPs is significantly larger than the respective sum for PTPs. As a consequence, expressions which predict that no transitions ever occur result in a high fitness. Instead, equal emphasis on PTPs and NTPs via a simple pre-processor heuristic was found to provide much better learning for TM.

The first step in this weight rebalance pre-processing is to generate two new time-series signals, $p_{k,n}$ and $n_{k,n}$, which decomposes the membership data into PTP and NTP components, respectively (Eq. 3.13-3.14). The $n_{k,n}$ signal is then scaled down by the ratio of the sum of the two components (Eq. 3.15), which ensures that the $\tilde{n}_{k,n}$ signal has equal influence on TM as the $p_{k,n}$ signal. Finally, the components are recombined to

produced the new weights, $\tilde{\gamma}_{k,n}$ (Eq. 3.16). This process is illustrated in Fig. 3.4.

$$p_{k,n} = \gamma_{k,n}(1 - \gamma_{k,n+1}) \quad (3.13)$$

$$n_{k,n} = \gamma_{k,n} - p_{k,n} \quad (3.14)$$

$$\tilde{n}_{k,n} = n_{k,n} \frac{\sum_{n=1}^{N-1} p_{k,n}}{\sum_{n=1}^{N-1} n_{k,n}} \quad (3.15)$$

$$\tilde{\gamma}_{k,n} = p_{k,n} + \tilde{n}_{k,n} \quad (3.16)$$

A benefit of this formulation is that it can be applied for uncertain or fuzzy membership values. To summarize, after the pre-processing for PTP-NTP weight rebalance described in Eq. 3.13-3.16, the search relationship in Eq. 3.17 and fitness metric in Eq. 3.18 is applied to TM for finding all transition events from mode k to mode k' . The best expression is selected using the AIC ranking based on the transition fitness.

$$\gamma_{k',n+1} = \text{sig}(t_{k \rightarrow k'}(\mathbf{u}_n)) \quad (3.17)$$

$$F_{\text{transition}} = - \frac{\sum_{n=1}^{N-1} \tilde{\gamma}_{k,n} \|\gamma_{k',n+1} - \text{sig}(t_{k \rightarrow k'}(\mathbf{u}_n))\|^2}{\sum_{n=1}^{N-1} \tilde{\gamma}_{k,n}} \quad (3.18)$$

The complete MMSR algorithm to learn analytic models of hybrid dynamical systems is summarized in Algorithm 3.3.

Algorithm 3.3: Multi-modal symbolic regression

```
1  input  → unclustered input-output data -  $\mathbf{u}_n, \mathbf{y}_n$ 
2          → the number of subfunctions -  $K$ 
3  output → behavior for each mode -  $f_k(\mathbf{u}_n)$ 
4          → variance for each mode -  $\sigma_k^2$ 
5          → transitions between each mode -  $t_{k \rightarrow k'}(\mathbf{u}_n)$ 
6
7  function symbolic_regression(search_relationship, fitness_function) :
8      initialize population with random expressions defined by search_relationship
9      for predefined computational effort :
10         generate new expressions from existing population (Fig.1.2)
11         calculate fitness of all expressions according to fitness_function
12         remove unsuitable expressions from the population
13         for each pop_expr in the population :
14             for each pareto_expr in the pareto_set :
15                 if ((pop_expr.fitness > pareto_expr.fitness) and
16                     (pop_expr.complexity <= pareto_expr.complexity)) :
17                     add pop_expression to pareto_set
18                     remove pareto_expression from pareto set
19     return pareto_set
20
21 # Clustered symbolic regression
22 initialize random membership values
23 for each behavior in  $K$  modes :
24     sr_solutions = symbolic_regression(Eq.3.6, Eq.3.7)
25     set behavior  $f_k$  to solution with lowest local AIC score in sr_solutions
26     set variance for each behavior -  $\sigma_k^2$  (Eq.3.5)
27 while convergence is not achieved :
28     for each behavior in  $K$  modes :
29         # expectation step
30         for all the  $N$  data points :
31             compute membership values -  $\gamma_{k,n}$  (Eq.3.4)
32         # maximization step
33         sr_solutions = symbolic_regression(Eq.3.6, Eq.3.7)
34         for each solution in sr_solutions :
35             compute temporary membership values -  $\check{\gamma}_{k,n}$  (Eq.3.4)
36             compute temporary variance -  $\check{\sigma}_k^2$  (Eq.3.5)
37             compute global fitness using temporary values -  $E_{CSR}$  (Eq.3.2)
38             compute AIC score using global fitness (Eq.3.8)
39             set behavior  $f_k$  to solution with lowest AIC score in sr_solutions
40             set variance to corresponding value -  $\sigma_k^2$  (Eq.3.5)
41 # Transition modelling
42 for each mode  $k$  in  $K$  modes :
43     for each different mode  $k'$  in  $K-1$  modes :
44         rebalance the PTP and NTP weights (Eq.3.13-3.16)
45         tm_solutions = symbolic_regression(Eq.3.17, Eq.3.18)
46         for each solution in tm_solutions :
47             compute AIC score using transition fitness (Eq.3.8)
48             set transition  $t_{k \rightarrow k'}(\mathbf{u}_n)$  to solution with lowest AIC score in tm_solutions
49
50 return behaviors  $f_k$ , variances  $\sigma_k^2$  and transitions  $t_{k \rightarrow k'}(\mathbf{u}_n)$ 
```

3.4 Results

This section begins with a description of the experimental setup for both the synthetic and real data experiments. Next is a discussion of the synthetic experiments, starting with an overview of alternative approaches, a list of the performance metrics, a summary of four data sets and finally, a discussion of MMSR performance in comparison to the baseline approaches. MMSR is then used to identify and characterize field-effect transistor modes, similar to those derived from first principles, based on real data. This section concludes with a brief discussion of the scalability of MMSR.

3.4.1 Experimental details

In these experiments, the publicly available Eureqa API [177] was used as a backend for the symbolic regression computation in both the CSR and TM. To illustrate the robustness of MMSR, the same learning parameters were applied across all the data sets, indicating that task-specific tuning of these parameters was not required:

- The SR for CSR was initially executed for 10000 generations and this upper limit was increased by 200 generations every iteration, until the global error produced less than 2% change for five EM iterations. Once CSR was complete, the SR for TM was a single 20000 generation search for each transition.
- The CSR algorithm was provided all the continuous inputs, while the TM algorithm was also provided with the one-hot encoding of binary signals, according to the data.
- The default settings in Eureqa, the SR backend, were used:
 - Population size = 64

- Mutation probability = 3%
- Crossover probability = 70%
- The basic algebraic building blocks were used for both algorithms: {constants, +, −, ×, /}. These building blocks were chosen as they form a fundamental set of basis operations that are capable of constructing more complex expressions. Additional building blocks such as trigonometric or transcendental functions could be included, but in their absence, numerical approximations, such as Taylor expansions, are inferred.

3.4.2 Synthetic data experiments

This section discusses a collection of experiments on hybrid systems generated by computer simulation. It begins with an introduction of alternative multi-modal model inference approaches, followed by an outline of the metrics used to measure their performance and a description of the data sets used for model comparison. This section concludes with a summary and discussion of the experimental results.

Alternative models

This subsection describes two traditional machine learning approaches to modeling multi-modal time-series data: fully recurrent neural networks and neural network based, input-output hidden Markov machines.

Fully Recurrent Neural Network—A fully Recurrent Neural Network (RNN) is a neural network where connections form a directed cycle (Fig. 3.5). This baseline recurrent network was composed of an input layer of nodes with linear transfer functions,

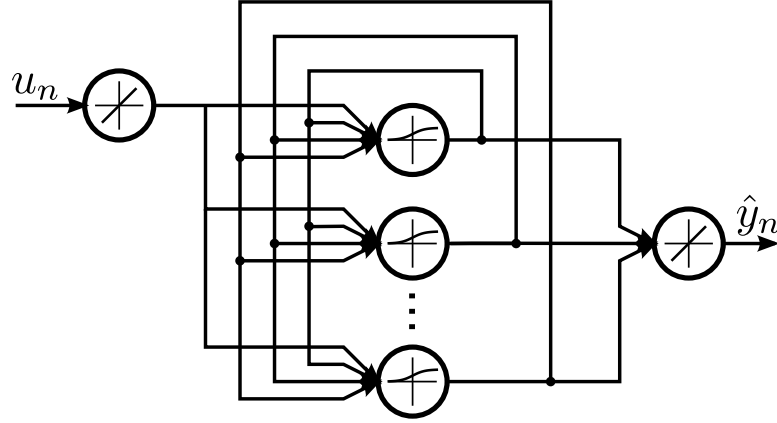


Figure 3.5: Schematic diagram of the fully recurrent neural network.

a single hidden layer of nodes with sigmoidal transfer functions and a linear transfer function as the output node. The output of the hidden layer was consequently fed back as an input with a one cycle delay, allowing the network to store memory and making it capable of modelling multi-modal behavior.

The network was implemented using the open source, machine learning library Py-Brain [170] and was trained via backpropagation through time [168]. The training data was split into a training and validation subset, where the training subset consists of the initial contiguous 75% portion of the data. The training was terminated either via early stopping or when the training error decreased by less than 0.01% for 10 iterations. The size of the hidden layers, h , ranged from 10, 25, 50 to 100 nodes based on complexity of the data set. The weights were initialized by sampling a zero mean Gaussian random variable with a standard deviation of 1. The learning rate was $\epsilon = 0.0005/h$ and used a momentum of 0.1. The learning rate was sufficiently small that the gradients never grew exponentially.

Neural Network Based IOHMM—The neural network based IOHMM (NNHMM) architecture by Bengio and Frasconi [9] is a Markov model that also captures input-output relationships. Provided with the number of modes, NNHMM uses two collec-

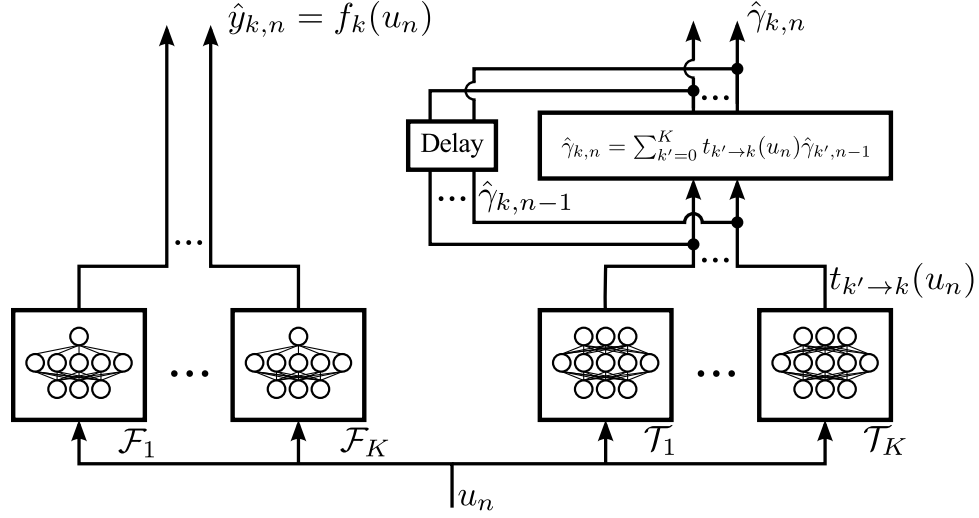


Figure 3.6: Schematic diagram of the neural network based IOHMM architecture [9]. \mathcal{F}_k are regression networks with a single sigmoidal hidden layer and \mathcal{T}_k are softmax networks with a single sigmoidal hidden layer.

tions of neural networks: one to predict the input-output mapping of each mode and another to predict the distribution of states. As no prior information was provided, the networks are designed to be as general as possible: a multilayer perceptron and one layer of hidden nodes with sigmoidal transfer functions. The input-output networks used linear input and output layers while the state prediction network used a linear input layer and a softmax output layer.

The Generalized EM algorithm (GEM) was applied for training was terminated either via early stopping or when the validation error produced less than a 0.01% decrease for 50 EM iterations. The size of the hidden layers, h , are identical for every network in the architecture and ranged from 5, 10, and 20 nodes. The weights were initialized by sampling a zero mean Gaussian random variable with a standard deviation of 1. The learning rate was $\epsilon = 0.0002/h$ with no momentum. The learning rate was sufficiently small that the gradients never grew exponentially.

Performance metrics

There are three performance metrics of interest: model accuracy, complexity and fidelity.

Model accuracy is a measure of ability of a learning algorithm to predict outputs given inputs. The trained model is used to predict the evolution of the time series data and the error is the negative log probability of a mixture of Laplacians (Eq. 3.1).

For time series prediction, the accumulation of the state error over time, also known as drift, becomes a significant factor. Repeated iterations of accurate but not-perfect transitions over a prolonged period of time will result in a significant accumulated error. Drift is managed with a closed-loop system (Fig. 3.7), where the output of the previous time step is also provided. With the MMSR algorithm, a closed-loop model is trivially constructed by setting the previous state probabilities according to the clustering component in CSR. However, the neural network based algorithms cannot be reformed into a closed-loop model without retraining the network or adapting the framework to execute some form of clustering.

Model complexity is a measure of the total number of free parameters required for the model. For MMSR, the complexity is dynamic and is measured as the sum of nodes in the expression trees. The neural network based algorithms have a static complexity, which is the number of hidden nodes in all of subnetworks. Although the node count does not account for the complexity of operations and more comprehensive measures exist [208], it does provide a simple and coarse measure of complexity and acts as a first approximation to human interpretability.

Model fidelity is a measure of the MMSR's ability to reproduce the form or mathematical structure of original system. This metric is important as it integral to the primary

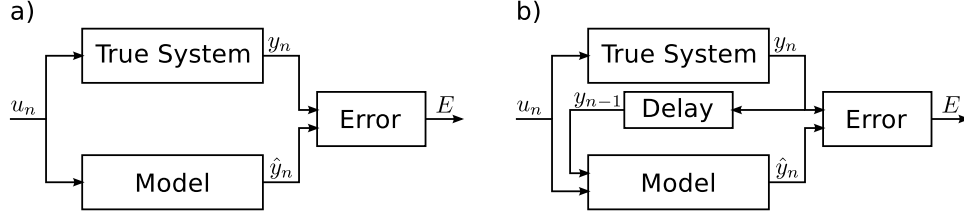


Figure 3.7: Schematic diagram of an open-loop and closed-loop system in a) and b), respectively.

goal of knowledge extraction – predictive accuracy is insufficient as the models must reproduce the expressions and not an approximation.

In symbolic representations, expressions are considered equivalent if and only if each subtree differs by at most scalar multiplicatives. For example, the expression $y = u^2/(1 + u)$ is considered to be equivalent to $y = 1.1u^2/(0.9 + u)$, but the Taylor series approximation about $u = 1 \rightarrow y = -0.125 + 0.5u + 0.125u^2$ is considered dissimilar regardless of its numeric accuracy. The fidelity is measured as the percentage of correctly inferred expression forms. In comparison, all neural network based systems are function approximations by design and thus, are immeasurable with respect to model fidelity.

Data Sets

Since there are no standardized data sets for the inference of hybrid dynamical systems, the MMSR algorithm was evaluated on a collection of four data sets based on classical hybrid systems [77, 205] and intelligent robotics [160]. These data sets range in complexity in both the discrete and continuous domains. Furthermore, these data sets contain non-trivial transitions and behaviors, and thus, present more challenging inference problems than the simple switching systems often used to evaluate parametric models of

Data Set	Mode (m_k)	Behavior (f_k)	No. of Points	Destination Mode ($m_{k'}$)	Transition ($t_{k \rightarrow k'}$)	No. of Transitions
Hysteresis Relay	1	$y = 1$	2037	2	$u > 0.5$	65
	2	$y = -1$	2059	1	$u < -0.5$	65
Continuous Hysteresis	1	$y = 0.5u^2 + u - 0.5$	2051	2	$u > 0.98$	40
	2	$y = -0.5u^2 + u + 0.5$	2045	1	$u < -0.98$	40
Phototactic Robot	1	$y = u_2 - u_1$	1568	2	$u_4 = 1$	36
				3	$u_5 = 1$	34
	2	$y = 1/(u_1 - u_2)$	1257	1	$u_3 = 1$	31
				3	$u_5 = 1$	40
	3	$y = 0$	1271	1	$u_3 = 1$	38
				2	$u_4 = 1$	35
Non-linear System	1	$y = u_1 u_2$	1302	3	$u_1^2 + u_2^2 < 9$	331
	2	$y = 6u_1/(6 + u_2)$	1535	1	$u_1^2 + u_2^2 > 25$	332
	3	$y = (u_1 + u_2)/(u_1 - u_2)$	1259	2	$u_1 u_2 > 0$	332

Table 3.1: Summary of test data sets for synthetic hybrid systems.

hybrid systems [109]. Simple switching systems have trivial discrete dynamics where the transition to any mode does not depend on the current mode.

Training and test sets were generated; the training sets were corrupted with varying levels of additive Gaussian noise, while the test sets remained noiseless. The level of noise was defined as the ratio of the Gaussian standard deviation to the standard deviation of the data set (Eq. 3.19). The noise was varied from 0% to 10% in 2% increments.

$$N_p = \frac{\sigma_{\text{noise}}}{\sigma_y} \quad (3.19)$$

The statistics of all four data sets are summarized in Table 3.1, while the system diagrams and test data set are shown in Figure 3.8.

Hysteresis Relay—The first data set is a hysteresis relay: a classical hybrid system [205, 207]. It is the fundamental component of hysteresis models and consists of two modes: ‘switched-on’ and ‘switched-off’. Each mode has a constant output and transitions occur at a threshold of the input. Although it is a simple hybrid dynami-

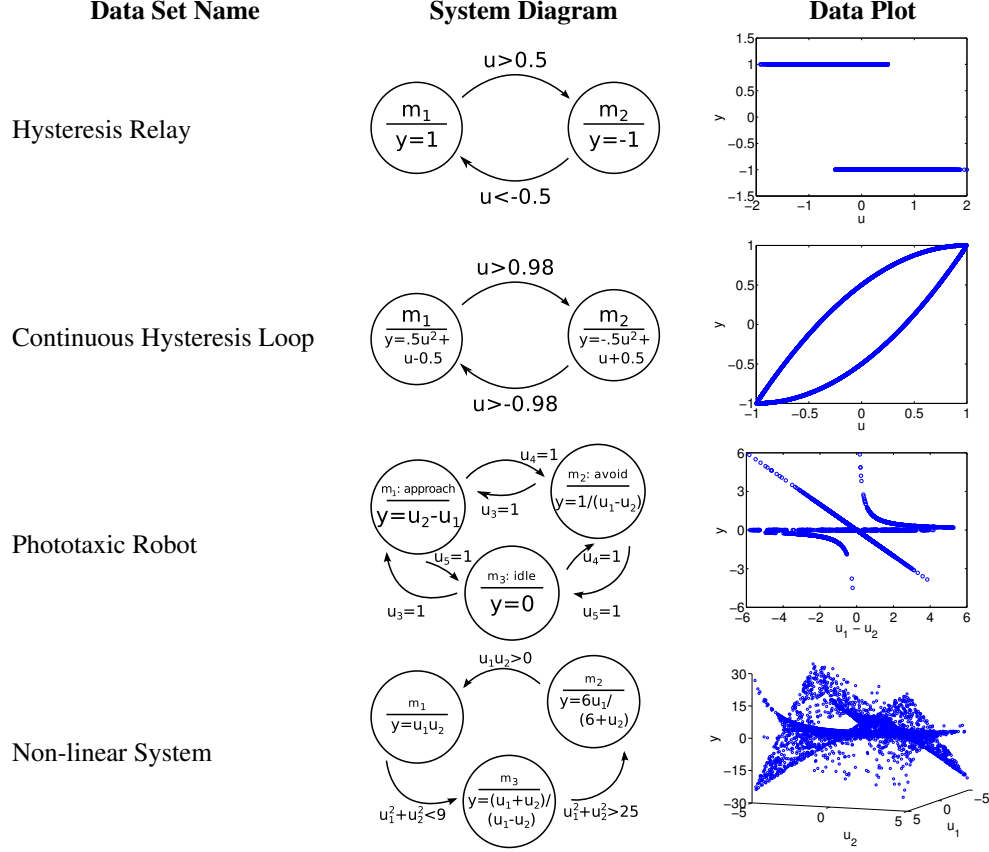


Figure 3.8: The system diagram and plots of the noiseless test data sets.

cal system with linear behaviors, it does not exhibit simple switching as the transitions depend on the mode since both behaviors are defined for $u \in [-0.5, 0.5]$.

Continuous Hysteresis Loop—The second data set is a continuous hysteresis loop: a non-linear extension of the classical hybrid system [207]. The Preisach model of hysteresis is used, where numerous hysteresis relays are connected in parallel and summed. As the number of hysteresis relays approaches infinity, a continuous loop is achieved. The data set is generated by repeatedly completing a single pass in the loop. Although there are still two modes, this data set is significantly more complex due to the symmetry of error functions about the line $y = u$, as well as the fact that transition depend on the mode and occur at a continuity in the output domain.

Phototaxic Robot—The third data set is a light-interacting robot [160]. The robot has phototaxic movement: it either approaches, avoids, or remains stationary depending on the color of light. The output y is velocity of the robot. There are five inputs: u_1 and u_2 are the absolute positions of robot and light, respectively, while $\{u_3, u_4, u_5\}$ is a binary, one-hot encoding of the light color, where 0 indicates the light is off and 1 indicates the light is on. This modeling problem is challenging due to the variety of inputs and non-uniform distribution of data. However, it does exhibit simple modal switching behavior that only depends on the light input.

Non-linear System—The fourth and final data set is a system without any physical counterpart, but the motivation for this system was to evaluate the capabilities of the learning algorithms for finding non-linear, symbolic expressions. The system consists of three modes, where all of the behaviors and transition conditions consist of non-linear equations which cannot be modeled via parametric regression without incorporating prior knowledge. All the expressions are a function of the variables u_1 and u_2 , the discriminant functions are not linearly separable and the transitions are modally dependent.

Experimental results

MMSR, along with the two parametric baselines, was evaluated on all four data sets and the performance metrics are summarized in Figure 3.9. This section begins with overview of the algorithms’ general performance, followed by case study analysis of each data set in the following subsections.

First, MMSR was able to reliably reconstruct the original model from the unlabeled, time-series data. The process of converting the program output into a hybrid

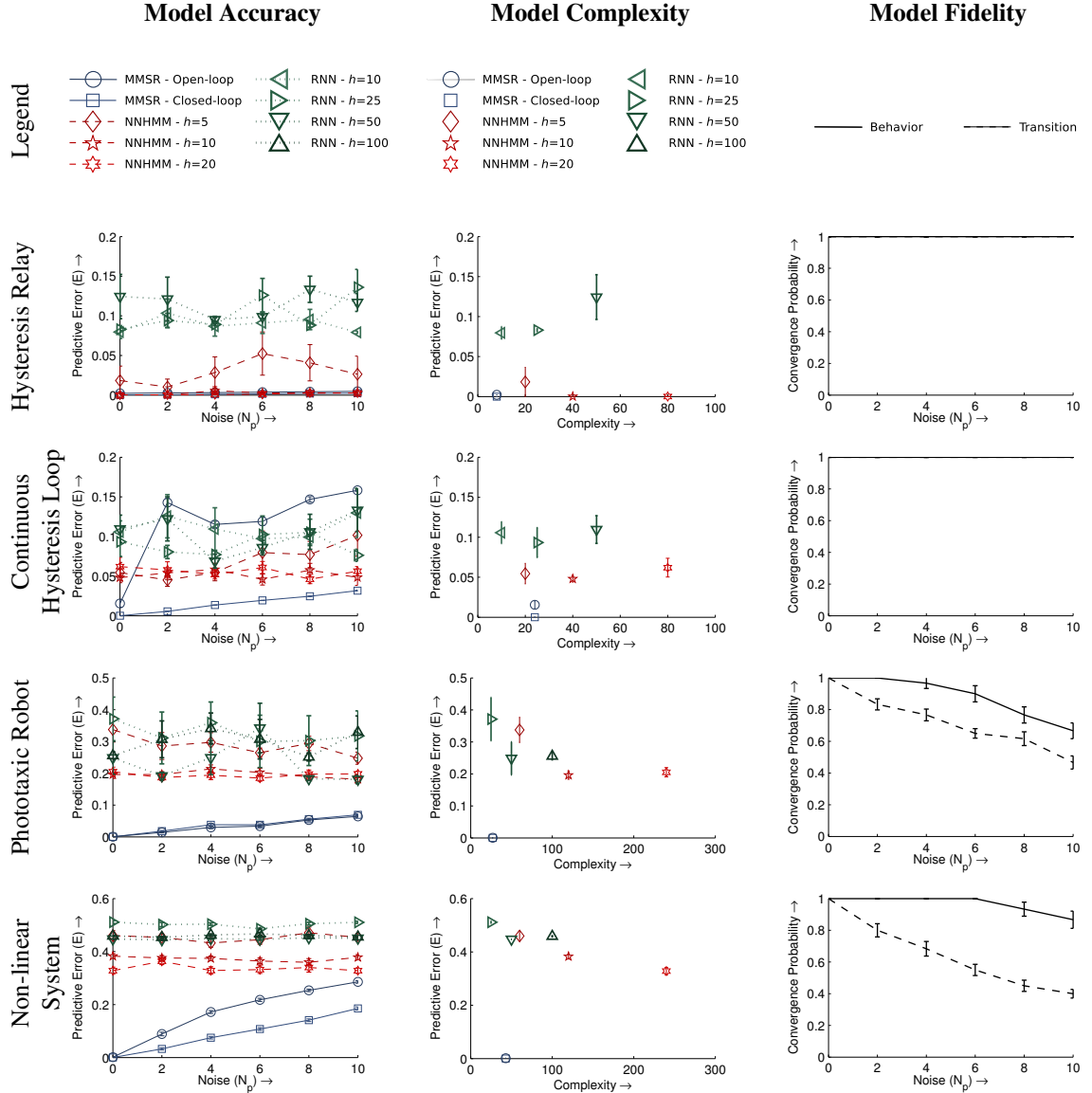


Figure 3.9: The performance metrics on the four synthetic hybrid systems. Error bars indicate standard error ($n = 10$).

automata model is summarized in Fig. 3.10, from a run obtained on the light-interacting robot training data with 10% noise. Provided with the number of modes, the algorithm searched for distinct behaviors and their subsequent transitions, returning a single symbolic expression for each of the inferred components. The expressions were algebraically simplified as necessary, and a hybrid dynamical model was constructed.

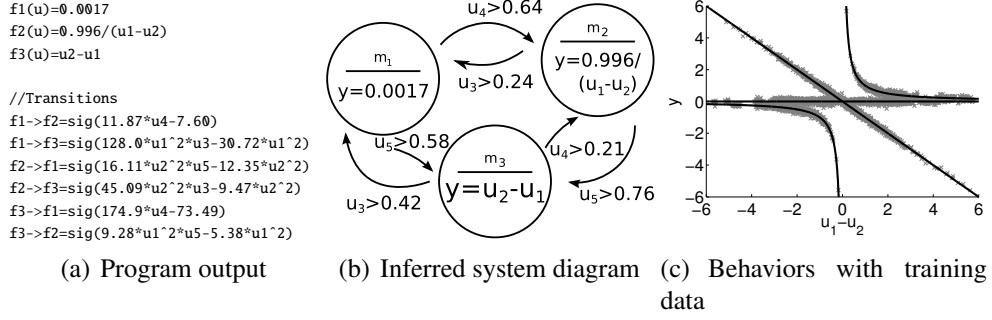


Figure 3.10: Conversion from program output to hybrid dynamical model for the phototactic robot with 10% noise. Algebraic simplifications were required to convert program output (a) to inequalities in canonical form (b).

Comparing the algorithms on predictive accuracy, the closed-loop MMSR model outperformed the neural network baselines on every data set across all the noise conditions. The open-loop MMSR model was able to achieve similar performance to its closed-loop counterpart for most systems, with the exception of the noisy continuous hysteresis loop. For low noise conditions, MMSR achieves almost perfect predictions, even in open-loop configurations.

In comparison, the RNN approach had difficulty modeling the time-series data sets while NNHMM performed marginally better. As the model accuracy is normalized by the standard deviation of the data set, these neural network baselines were able to capture some characteristics of the data set and performed much better than predicting the mean of the data, which would achieve an error of 1. However, other than the simplest data set, none of the parametric approaches were able to converge on an accurate representation, even with noiseless training data.

There is an inverse relationship between the generality of the algorithm and its performance at inferring hybrid dynamical systems. Although RNNs are capable of representing a wide variety of phenomena, the learning algorithm often settles on a poor

local optimum while NNHMM leverages a structural composition to achieve marginally better performance. MMSR, however, is tailored to inferring hybrid dynamical systems from unlabeled, time-series data and consequently infers a superior model for numerical predictions.

Furthermore, not only was MMSR a superior predictor, the numerical accuracy was achieved with less free parameters than the neural network baselines. Even though the measure of counting nodes provides only a coarse measure of complexity, the neural network approaches have significantly more error despite having up to five times the number of free parameters on noiseless training data. This suggests that the symbolic approach is better suited for the primary goal of knowledge extraction, by providing accurate as well as parsimonious models.

In addition, for the neural network approaches, increasing the model complexity does not necessarily result in greater accuracy. In fact, for most data sets, once the number of hidden nodes reached a threshold, the trained models generally become less accurate despite having additional modelling capabilities. For multi-model problems, the parameter space is non-convex and contains local optima – as the number of hidden nodes increases, the probability of finding a local optima increases as well. Thus, for parametric models, the number of hidden nodes must be tuned to account for the complexity of the data set, presenting another challenge to the arbitrary application of parametric models.

Finally, MMSR was able to achieve reliable model fidelity. In the noiseless training sets, the correct expressions for both the behaviors and events were inferred with perfect reliability. As the signal to noise ratio was increased, the probability of convergence varied significantly depending on the characteristics of the data set. Generally, the algorithm was able to repeatedly find the correct form for the behaviors for the majority of

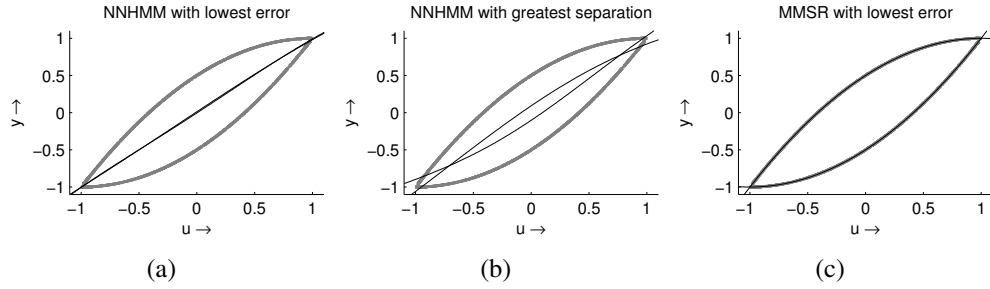


Figure 3.11: The input-output relationship of the regression networks of NNHMM and symbolic expressions of MMSR (black) overlaid on the Continuous Hysteresis Loop data (grey).

the data sets. In contrast, the transition expressions were more difficult to infer since the model fidelity deteriorates at lower noise levels. This is a result of TM's dependence on accurate membership values from CSR – noisy data leads to larger classification errors, amplifying the challenge of modeling transitions.

Despite the model fidelity's sensitivity to noise, the algorithm was nonetheless able to accurately predict outputs for a wide range of noise conditions. The inferred expressions, regardless of the expression fidelity, were still accurate numerical approximations for both open- and closed-loop models.

Hysteresis Relay—This simple data set was modeled by accurately by both MMSR and NNHMM, while RNN had relative difficulties. This was the only data set that NNHMM was able to achieve near perfect accuracy with ten or more hidden nodes per network, but failed when provided with only five hidden nodes per network. In terms of model fidelity, MMSR was able to achieve perfect expressions with respect to all the noise conditions.

Continuous Hysteresis Loop—This data set was ideal as it was sufficiently difficult to model, but simple enough to analyze and provide insight into how the algorithms per-

form on hybrid dynamical systems. The closed-loop MMSR was able to significantly outperform NNHMM and RNN under all noise conditions, but the open-loop MMSR fared worse than the parametric baselines in the presence of noise. This result was particularly interesting, since perfect model fidelity was achieved for all noise conditions. The predictive error in the open-loop MMSR occurred as a result of the continuous transition condition – under noisy conditions, the model can fail to predict a transition even with a correct model. As a result, a missed transition accumulates significant error for open-loop models. A closed-loop model is able to account for missed transitions, resulting in consistently accurate models.

Next, NNHMM outputs were analyzed to understand the discrepancy in predictive accuracy. Fig. 3.11 shows the input-output relationships of NNHMM’s best performing model, NNHMM’s model that obtains the greatest separation and MMSR’s best performing model, respectively. NNHMM had significant difficulties breaking the symmetry in the data set as the best model captured only the symmetry, while the locally-optimal asymmetrical model was both inferior in predictive accuracy and was significantly far from the ground truth. In comparison, MMSR was able to deal with the symmetrical data and infer unique representations. Such analysis could not be applied to RNNs as it is impossible to decouple the input-output relationships from the model transition components.

Phototaxic Robot—The phototaxic robot provided a challenging problem with an increased number of modes and asymptotic behaviors. Also, the distribution of the data was non-uniform and deceptive as it was sparse around the non-linear features. However, MMSR was able to achieve perfect model fidelity for low noise systems, which slowly degraded with respect to noise. Compared to the neural network approaches, both the open-loop and closed-loop produced significantly more accurate predictions under

every noise condition. Note the simple switching behavior resulted in open-loop model accuracy that is comparable to the closed-loop counterpart, suggesting that closed-loop models are not necessary for simple switching systems.

This data set provides an example of how symbolic expressions aid in knowledge abstraction as it is easy to infer that the relative distance between the robot and the light position, $u_1 - u_2$, is an integral component of the system as it is a repeated motif in the each of the behaviors. It is significantly more difficult to extract the same information from parametric approaches like neural networks.

Non-linear System—The final data set provided a difficult modelling challenge that included non-linear behaviors which cannot be modeled by by parametric regression. Yet, MMSR reliably inferred the correct model for low noise systems and produced accurate predictions in all noise levels despite the noise sensitivity of model fidelity. The neural network approaches were significantly less accurate while using more free parameters.

3.4.3 Real data experiment

This section provides a case study of MMSR on real-world data while also exemplifying the benefits of symbolic model inference. This case study involves the inference of an n-channel Metal-Oxide Semiconductor Field-Effect Transistor (nMOSFET), a popular type of transistor ubiquitous in digital and analog circuits. nMOSFETs exhibit three distinct characteristics, which are governed by the physical layout and the underlying physics [179], making them an ideal candidate for hybrid system analysis.

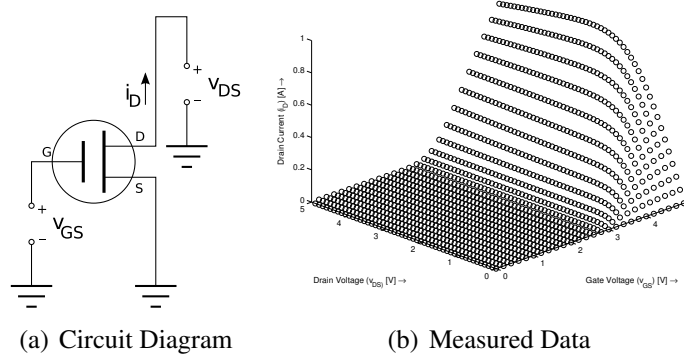
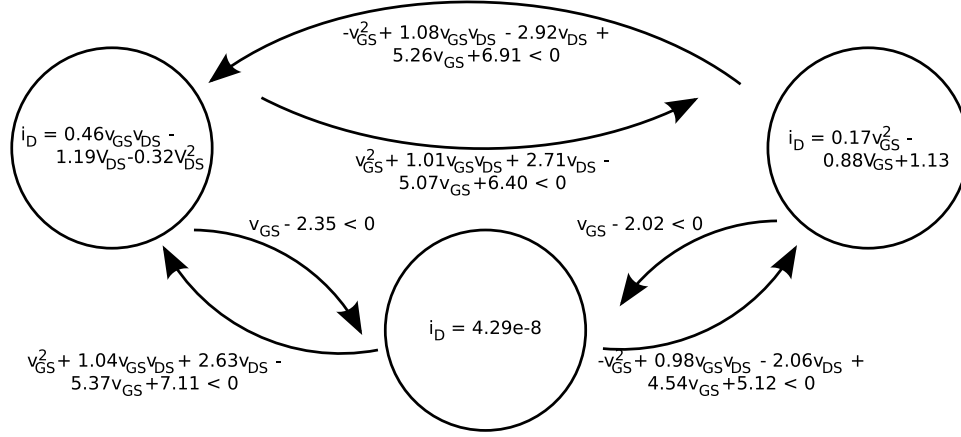


Figure 3.12: A circuit diagram indicating the two input voltages, v_{GS} and v_{DS} , and the output current i_D , and the measured 3D data plot from the ZVNL4206AV nMOSFET.

The transistor was placed in a standard configuration to measure the current-voltage characteristics, where the drain current is set as a function of the gate and drain voltages (Fig. 3.12a). The transistor was a Diodes Inc. ZVNL4206AV nMOSFET and the data was recorded with a Keithley 2400 general purpose sourcemeter. The data was collected via random voltage sweeps from 0-5V, and the subsequent current was measured (Fig. 3.12b).

The three discrete modes as well as the two-dimensional, non-linear input-output mapping makes this a non-trivial modelling problem. Furthermore, the regions are non-overlapping and continuous, which add another challenge in discerning the discrete modes. After applying MMSR with the setup described in Section 3.4.1, a hybrid dynamical system was inferred (Fig. 3.13a). MMSR was applied for ten independent runs and the median performing model was reported. As the transitions events were consistent between modes, which is indicative of the simple switching behavior exhibited by transistors, the system diagram was simplified to a piecewise representation with additional symbolic manipulations (Fig. 3.13b).



(a) Inferred system diagram

$$i_D = \begin{cases} 4.29e-8 & , \quad \text{if } v_{GS} \leq 2.02 \\ 0.46((v_{GS} - 2.59)v_{DS} - 0.71v_{DS}^2) & , \quad \text{if } v_{GS} > 2.68 \text{ and } (v_{GS} - 1.01v_{DS}) > 2.39 \\ 0.17(v_{GS} - 2.76)(v_{GS} - 2.40) & , \quad \text{if } v_{GS} > 2.11 \text{ and } (v_{GS} - 0.98v_{DS}) \leq 2.43 \end{cases}$$

(b) Inferred mode expressions

$$i_D = \begin{cases} 0 & , \quad \text{if } v_{GS} \leq k_1 \\ k_2((v_{GS} - k_1)v_{DS} - \frac{1}{2}v_{DS}^2) & , \quad \text{if } v_{GS} > k_1 \text{ and } (v_{GS} - v_{DS}) > k_1 \\ \frac{1}{2}k_2(v_{GS} - k_1)^2 & , \quad \text{if } v_{GS} > k_1 \text{ and } (v_{GS} - v_{DS}) \leq k_1 \end{cases}$$

(c) Classically derived mode expressions

Figure 3.13: The inferred hybrid model compared to the derived expressions.

When the inferred expressions are compared to classical equations [179], the results are remarkably similar (Fig. 3.13c). This suggests that MMSR is capable of inferring the ground truth of non-trivial systems from real-world data. While the model is sufficiently numerically accurate, the more impressive and relevant consequence is that MMSR was able to find the same expressions as engineer would derive from first principles, but inferred the results from unlabeled data. For an engineer or scientist presented with an unknown device with multi-modal behavior, beginning with apt, mathematical descriptions of a system might provide essential insight and understanding to determining the governing principles of that system. This capability provides an important advantage over traditional parametric machine learning models.

3.4.4 Scalability

Given that extracting dynamical equations from experimental data is an NP hard problem [40], determining the optimal model for hybrid dynamical systems is intractable. While evolutionary computational approaches are heuristic, exploratory methods that unable to guarantee optimality of a candidate model, in practice, they often find good and meaningful solutions. Rather than a traditional lower-bound analysis, analyzing the computational complexity is used to provide insight to the scope of problems that are well suited for MMSR inference.

To assess the performance scalability of MMSR, the computational complexity of SR must first be analyzed as it is the primary computational kernel. As convergence on the global solution is not guaranteed, in the worst-case analysis, the complete search space is exhausted in a stochastic manner. For b building blocks and a tree depth size of c nodes, the search space grows exponentially with a complexity of $O(b^c)$. However, on average, SR performs significantly better than the worst case, although the performance is highly case dependent. Furthermore, evolutionary algorithms are naturally parallel, providing scalability with respect to the number of processors.

For the MMSR learning algorithm, two components are analyzed independently. With the worst-case SR complexity $O(b^c)$ and k modes, CSR has a compounded linear complexity with respect to the number of modes, $O(kb^c)$, while TM has a quadratic complexity of $O(k^2b^c)$, since transitions for every combination of modes must be considered. In terms of worst-case computational effort, this suggests that this algorithm would scale better for systems with numerous simple modes than it would for systems with fewer modes of higher complexity. For the data sets described in this section, the algorithm required an average of 10 and 45 minutes for the bi- and tri-modal systems, respectively, on a single core of a 2.8GHz Intel processor.

3.5 Conclusion and future work

A novel algorithm, Multi-Modal Symbolic Regression (MMSR), was presented to infer non-linear, symbolic models of hybrid dynamical systems. MMSR is composed of two general subalgorithms. The first subalgorithm is Clustered Symbolic Regression (CSR), designed to construct expressions for piecewise functions of unlabeled data. By combining Symbolic Regression (SR) with Expectation-Maximization (EM), CSR is able to separate the data into distinct clusters, and then subsequently find mathematical expressions for each subfunction. CSR exploits the Pareto front of SR to consistently avoid locally optimal solutions, a common challenge in EM mixture models. The second subalgorithm is Transition Modeling (TM), which searches for binary classification boundaries and expresses them as a symbolic inequality. TM uniquely capitalizes on the pre-existing SR infrastructure through function composition. These two subalgorithms are combined and used to infer symbolic models of hybrid dynamical systems.

MMSR is applied to four synthetic data sets, which span a range of classical hybrid automata and intelligent robotics. The training data was also corrupted with various levels of noise. The inferred models were compared via three performance metrics: model accuracy, complexity, and fidelity. MMSR inferred reliable models for noiseless data sets and outperformed its neural network counterparts in both model accuracy as well as model complexity. Furthermore, MMSR was used to identify and characterize field-effect transistor modes, similar to those derived from first principles, demonstrating a possible real-world application unique to this algorithm.

Symbolic modelling provides numerous benefits over parametric numerical models with the primary advantage of operating in symbolic expressions, the standard language of mathematics and science. Symbolic modelling provides the potential for

knowledge abstraction and deeper understanding, as compared to the alternative of numeric, parametric approaches. In addition, there is a wealth of theory in symbolic mathematics, including approximation and equivalence theories such as Taylor expansions, which may aid understanding inferred models. Even having symbolic expressions to identify reoccurring motifs and subexpressions may provide insight in the inner workings of the system.

A primary concern for symbolic modeling is how well it extends as the complexity increases and whether an easily interpretable model exists. However, the alternatives struggle equally in such cases. Deriving models from first principles is often similarly challenging while parametric approaches, such as RNN and NNHMM, are likely to settle on local optima and have difficulty achieve even numerically accurate models, even for relatively simple hybrid dynamical systems.

This work is the first step towards the generalized problem of modeling complex, multi-modal dynamical systems. While symbolic expressions may not exist for complex systems, it does present a viable alternative approach that may have the additional benefit of insight and interpretability. Future work includes extending the model to infer differential equations and investigating higher dimensional systems.

CHAPTER 4

UNCOVERING HIDDEN DYNAMICAL VARIABLES

This chapter focuses on a collection of algorithms to discover hidden dynamical variables from time-series data. The first section discusses space space transformations of dynamical systems, which is used as the theoretical foundation for the remainder of the chapter. The second section discusses an algorithm to infer ordinary differential equations of arbitrary order while the third section presents an algorithm for discovering hidden variables using a principle of simplicity.

4.1 State space transformations of dynamical systems

This section presents the theoretical foundation for uncovering hidden dynamical variables from time-series data. At its core, this theory revolves around state space transformations of dynamical systems. Sections 4.1.1 and 4.1.3 discuss the theory for transformations of nonlinear dynamical systems with one and multiple observed variables, respectively, while Sections 4.1.2 and 4.1.4 provide examples for the one and multiple observed variables, using the Lotka-Volterra population dynamics and Chua circuit, respectively.

4.1.1 Transformations of nonlinear dynamical systems with one observed variable

This section provides the mathematical basis for the discovering hidden variables in nonlinear dynamical systems from time-series data. Many of the concepts in this section have been previously outlined in the field of transformations of nonlinear dynamical

systems [50], but the following four sections attempts to formalize these concepts within a hidden state variable framework.

It begins with a definition of nonlinear dynamical systems with one observed variable followed by a discussion of the necessary conditions for which a system can be transformed arbitrarily while maintaining equivalent observations. Next, the controllable canonical realization is defined and as well as properties such as its relationship to higher-order ordinary differential equations of a single variable, and the transformations to and from arbitrary systems. Finally, the entire mathematical framework is discussed in the context of finding hidden variables.

Definition 4.1.1. *Nonlinear dynamical system with one observed variable.* A nonlinear dynamical system is defined a set of coupled, nonlinear, autonomous Ordinary Differential Equations (ODEs):

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}) \\ \mathbf{y} &= \mathbf{g}(\mathbf{x})\end{aligned}\tag{4.1}$$

where $\mathbf{x} = [x_1 \dots x_n]^T \in \mathbb{R}^n$ are the state variables, $\dot{\mathbf{x}} \in \mathbb{R}^n$ is the first-order time derivative of the state variables, $\mathbf{y} = [y_1 \dots y_m]^T \in \mathbb{R}^m$ are the observed or measured variables, $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ defines the smooth state transition function, and $\mathbf{g} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ defines the output relationship. The system is assumed to be a minimal realization, and thus is both controllable and observable.

Since the focus is on discovering the state variables, the analysis can be restricted to systems where the observed variables are direct measurements of the state variables. As a result, the output relationship is constrained to a Boolean matrix where there is only one non-zero element per row: $\mathbf{g}(\mathbf{x}) = \mathbf{C}\mathbf{x}$, $\mathbf{C} \in \mathbb{B}^{m \times n}$.

Hidden variables are the state variables that are not represented by the observed variables. For a system with hidden variables, the number of observed variables must be strictly less than the number of state variables, or $m < n$.

In this section, the analysis will be restricted to systems with only one variable is observed, or $m = 1$. Since the state variables can be rearranged arbitrarily, the observed variable is defined as the first state variable without any loss of generality. Thus, a nonlinear dynamical system with one observed variable is defined as:

$$\begin{aligned}\dot{x}_1 &= f_1(x_1, \dots, x_n) \\ &\vdots \\ \dot{x}_n &= f_n(x_1, \dots, x_n) \\ y &= x_1\end{aligned}\tag{4.2}$$

Given a system with one observed variable, the remaining unobserved state variables can be transformed arbitrarily while maintaining an identical output of the system. An alternate realization of the state space, $\tilde{\mathbf{x}}$, is defined through the nonlinear transformation:

$$\begin{aligned}\tilde{x}_1 &= T_1(x_1, \dots, x_n) = x_1 \\ \tilde{x}_2 &= T_2(x_1, \dots, x_n) \\ &\vdots \\ \tilde{x}_n &= T_n(x_1, \dots, x_n)\end{aligned}\tag{4.3}$$

where the $\mathbf{T} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is nontrivial bijective map.

Theorem 4.1.2. Existence of equivalent transformations. *For any nontrivial bijective transformation $\tilde{\mathbf{x}} = \mathbf{T}(\mathbf{x})$, there exists an alternate realization of the dynamical system with a different state space that has equivalent observed variables.*

Proof. By definition, the observed variable in both systems are identical, $\tilde{x}_1 = x_1$. To prove the existence of an alternate realization with a different state space, one only needs to determine a closed form expression of the state transition function that corresponds to the transformed state, $\dot{\tilde{x}}_i = \tilde{f}_i(\tilde{x}_1, \dots, \tilde{x}_n), \forall i \in \{1, \dots, n\}$. By applying the chain rule and using the inverse transformation for each state variable, $x_i = \tilde{T}_i^{-1}(\tilde{x}_1, \dots, \tilde{x}_n), \forall i \in \{1, \dots, n\}$, differentiating the transformed state variables with respect to time obtains:

$$\begin{aligned}
\dot{\tilde{x}}_i &= \sum_{j=1}^n \frac{\partial T_i}{\partial x_j}(x_1, \dots, x_n) \cdot \dot{x}_j \\
&= \sum_{j=1}^n \frac{\partial T_i}{\partial x_j}(x_1, \dots, x_n) \cdot f_j(x_1, \dots, x_n) \\
&= \sum_{j=1}^n \frac{\partial T_i}{\partial x_j}(x_1, \dots, x_n) \cdot f_j(x_1, \dots, x_n) \Big|_{x_i = \tilde{T}_i^{-1}(\tilde{x}_1, \dots, \tilde{x}_n)} \\
&= \tilde{f}_i(\tilde{x}_1, \dots, \tilde{x}_n)
\end{aligned} \tag{4.4}$$

which provides a closed form expression of the alternate state transition function. \square

Definition 4.1.3. Controllable canonical realization. *The controllable canonical form or realization, with a state space notation $\hat{\mathbf{x}}$, is defined as follows:*

$$\begin{aligned}
\dot{\hat{x}}_1 &= \hat{x}_2 \\
&\vdots \\
\dot{\hat{x}}_{n-1} &= \hat{x}_n \\
\dot{\hat{x}}_n &= \hat{f}(\hat{x}_1, \dots, \hat{x}_n)
\end{aligned} \tag{4.5}$$

The controllable canonical realization has a history in control theory and is named due to its property that its state-space realization is guaranteed to be controllable if the $\dot{\hat{x}}_n$ state transition function has an input [140]. This realization is also popular in the domain of numerical methods, where it is used to transform an n^{th} -order ODE into a system of n coupled, first-order ODEs which can be solved by iterative methods.

Definition 4.1.4. Higher-order ordinary differential equation. A higher-order ODE is an ODE with an order greater than one.

Corollary 4.1.5. Higher-order ordinary differential equation realization. Every controllable canonical realization has an equivalent higher-order ordinary differential equation of a single variable: $\frac{d^n x_1}{dt^n} = F\left(x_1, \dot{x}_1, \dots, \frac{d^{n-1} x_1}{dt^{n-1}}\right)$.

Proof. Note that each state variable is a higher-order time derivative of the observed variable. Converting one realization into the other consists of changing the state variable notation. □

The relationship between the controllable canonical realization and the higher-order ODE realization highlights two critical properties that are leveraged in the search for hidden variables. First, since a controllable canonical realization exists for any nontrivial bijective transformations (Theorem 4.1.6), a higher-order ODE realization exists and inferring it is equivalent to inferring a state-space realization. Second, the controllable canonical realization is a minimal realization, which defines the necessary and sufficient number of state variables in the system. Thus, inferring a higher-order ODE realization of the system defines the number of state variables as well as the number of hidden variables.

Theorem 4.1.6. Transformation to the controllable canonical realization. If the following transformation, $\hat{\mathbf{x}} = \mathbf{T}(\mathbf{x})$, is bijective:

$$\begin{aligned}\hat{x}_1 &= T_1(x_1, \dots, x_n) = x_1 \\ \hat{x}_2 &= T_2(x_1, \dots, x_n) = \frac{dx_1}{dt} = f_1(x_1, \dots, x_n) \\ &\vdots \\ \hat{x}_n &= T_n(x_1, \dots, x_n) = \frac{d^{n-1} x_1}{dt^{n-1}} = \frac{d\hat{x}_{n-1}}{dt} = \sum_{j=1}^n \frac{\partial T_i}{\partial x_j}(x_1, \dots, x_n) \cdot f_j(x_1, \dots, x_n)\end{aligned}\tag{4.6}$$

then there exists a controllable canonical realization of the system.

Proof. Given that the transformation $\hat{\mathbf{x}} = \mathbf{T}(\mathbf{x})$ is bijective, it satisfies the conditions for Theorem 4.1.2. One only needs to show that the transformation results in the controllable canonical realization. Differentiating the transformation obtains:

$$\begin{aligned}
\dot{\hat{x}}_1 &= \dot{x}_1 = f_1(x_1, \dots, x_n) = \hat{x}_2 \\
&\vdots \\
\dot{\hat{x}}_{n-1} &= \frac{d\hat{x}_{n-1}}{dt} = \frac{d}{dt} \left(\frac{d^{n-2}x_1}{dt^{n-2}} \right) = \frac{d^{n-1}x_1}{dt^{n-1}} = \hat{x}_n \\
\dot{\hat{x}}_n &= \frac{d\hat{x}_n}{dt} = \sum_{j=1}^n \frac{\partial T_i}{\partial x_j}(x_1, \dots, x_n) \cdot f_j(x_1, \dots, x_n) \Big|_{x_i=T_i^{-1}(\hat{x}_1, \dots, \hat{x}_n)} = \hat{f}(\hat{x}_1, \dots, \hat{x}_n)
\end{aligned} \tag{4.7}$$

which is the controllable canonical realization. \square

Theorem 4.1.7. Transformation from the controllable canonical realization. For a transformation $\hat{\mathbf{x}} = \mathbf{T}(\tilde{\mathbf{x}})$, if the Jacobian determinant of the transformation is non-zero, then the realization for the state space $\tilde{\mathbf{x}}$ can be determined from the controllable canonical realization.

Proof. First, controllable canonical realization and the transformation can be combined to define the following relationship:

$$\begin{aligned}
\dot{\hat{x}}_1 &= \hat{x}_2 = T_2(\tilde{x}_1, \dots, \tilde{x}_n) \\
&\vdots \\
\dot{\hat{x}}_{n-1} &= \hat{x}_n = T_n(\tilde{x}_1, \dots, \tilde{x}_n) \\
\dot{\hat{x}}_n &= \hat{f}(T_1(\tilde{x}_1, \dots, \tilde{x}_n), \dots, T_n(\tilde{x}_1, \dots, \tilde{x}_n))
\end{aligned} \tag{4.8}$$

Next, differentiating $\hat{\mathbf{x}} = \mathbf{T}(\tilde{\mathbf{x}})$ with respect to time by applying the chain rule obtains:

$$\dot{\hat{\mathbf{x}}} = \mathbf{J}_T(\tilde{\mathbf{x}}) \cdot \dot{\tilde{\mathbf{x}}} \quad (4.9)$$

where $\mathbf{J}_T(\tilde{\mathbf{x}})$ is the Jacobian matrix of $\mathbf{T}(\tilde{\mathbf{x}})$.

Since the Jacobian determinant of $\mathbf{T}(\tilde{\mathbf{x}})$ is non-zero and the inverse exists, Eqs. 4.8 and 4.9 can be combined to obtain a closed-form expression of the state space realization:

$$\begin{aligned} \dot{\tilde{\mathbf{x}}} &= \mathbf{J}_T^{-1}(\tilde{\mathbf{x}}) \cdot \dot{\hat{\mathbf{x}}} \\ &= \mathbf{J}_T^{-1}(\tilde{\mathbf{x}}) \cdot \begin{bmatrix} T_2(\tilde{x}_1, \dots, \tilde{x}_n) \\ \vdots \\ T_n(\tilde{x}_1, \dots, \tilde{x}_n) \\ \hat{f}(T_1(\tilde{x}_1, \dots, \tilde{x}_n), \dots, T_n(\tilde{x}_1, \dots, \tilde{x}_n)) \end{bmatrix} \end{aligned} \quad (4.10)$$

which proves the existence of the realization of $\tilde{\mathbf{x}}$. \square

Note, unlike Theorem 4.1.2 and Theorem 4.1.6, Theorem 4.1.7 does not require that the inverse transformations are determined or that they even exist. This unique property arises from the fact that the transformed state variables are the argument of the transformation as opposed to the result of the transformation. Although the resulting state space may not be bijective, the condition that the Jacobian determinant of the transformation is non-zero is a significantly more relaxed constraint, which makes it easier to generate arbitrary candidate transformations and for algorithms to search for hidden variables. For a candidate transformation, finding the realization from the controllable canonical realization only requires differentiation, computing a determinant, matrix inversion and matrix multiplication as opposed to significantly more challenging task of solving the inverse relations for coupled, nonlinear expressions.

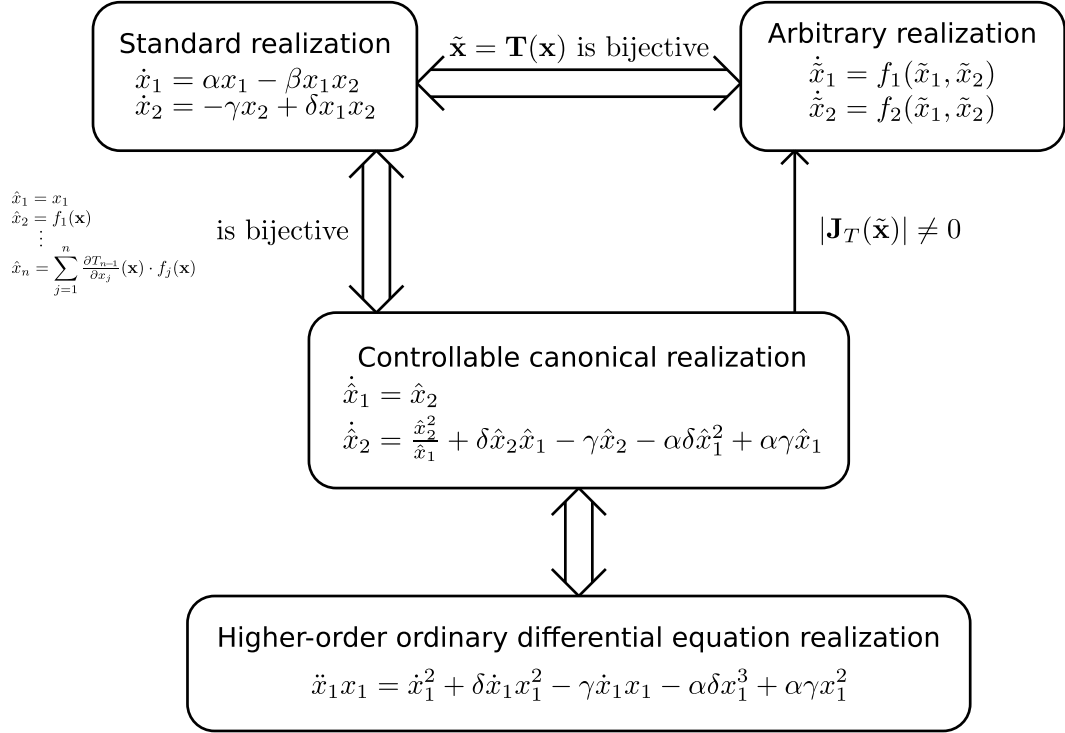


Figure 4.1: A flowchart outlining the four different realizations and conditions for transforming between them, using the Lotka-Volterra dynamics as an example (Section 4.1.2).

The relationship between the different realizations, as well as the conditions required for their transformations, is described in Fig. 4.1.

The approach to discovering hidden variables from time series data is then defined by two search algorithms. The first algorithm infers ODE models from the measured time series data. Once a candidate ODE model is obtained, the model is converted to its corresponding controller canonical realization using Corollary 4.1.5. The second algorithm searches for parsimonious transformations by inferring candidate state space realizations from the controller canonical realization using Theorem 4.1.7.

4.1.2 Example realizations for the Lotka-Volterra population dynamics system

To illustrate the process of discovering hidden variables from a nonlinear dynamical system with a one observed variable, this section provides an example using the Lotka-Volterra population dynamics [209]. This system is a second-order, nonlinear dynamical system that models the dynamics of predator-prey interactions. The standard realization of the Lotka-Volterra system is:

$$\begin{aligned}\dot{x}_1 &= \alpha x_1 - \beta x_1 x_2 \\ \dot{x}_2 &= -\gamma x_1 + \delta x_1 x_2\end{aligned}\tag{4.11}$$

where x_1 and x_2 represents the respective populations of predator and prey species (traditionally lynxes and hares), and $\alpha, \beta, \gamma, \delta > 0$ are all model parameters.

The analysis begins by deriving the controller canonical realization of the system. As per Theorem 4.1.6, the transformation that will lead to the controller canonical realization is:

$$\begin{aligned}\hat{x}_1 &= T_1(x_1, x_2) = x_1 \\ \hat{x}_2 &= T_2(x_1, x_2) = \alpha x_1 - \beta x_1 x_2\end{aligned}\tag{4.12}$$

Next, the inverse transformation can be determined via simple algebraic manipulation:

$$\begin{aligned}x_1 &= T_1^{-1}(\hat{x}_1, \hat{x}_2) = \hat{x}_1 \\ x_2 &= T_2^{-1}(\hat{x}_1, \hat{x}_2) = \frac{\alpha \hat{x}_1 - \hat{x}_2}{\beta \hat{x}_1}\end{aligned}\tag{4.13}$$

Following Eq. 4.6, the controller canonical realization can be determined:

$$\begin{aligned}
\dot{\hat{x}}_1 &= \frac{\partial T_1}{\partial x_1} \dot{x}_1 + \frac{\partial T_1}{\partial x_2} \dot{x}_2 \Big|_{x_i=T_i^{-1}(\hat{x}_1, \dots, \hat{x}_n)} \\
&= \alpha x_1 - \beta x_1 x_2 \Big|_{x_i=T_i^{-1}(\hat{x}_1, \dots, \hat{x}_n)} \\
&= \alpha \hat{x}_1 - \beta \hat{x}_1 \left(\frac{\alpha \hat{x}_1 - \hat{x}_2}{\beta \hat{x}_1} \right) \\
&= \hat{x}_2
\end{aligned} \tag{4.14}$$

$$\begin{aligned}
\dot{\hat{x}}_2 &= \frac{\partial T_2}{\partial x_1} \dot{x}_1 + \frac{\partial T_2}{\partial x_2} \dot{x}_2 \Big|_{x_i=T_i^{-1}(\hat{x}_1, \dots, \hat{x}_n)} \\
&= (\alpha - \beta x_2)(\alpha - \beta x_1 x_2) + (-\beta x_1)(\gamma x_1 + \delta x_1 x_2) \Big|_{x_i=T_i^{-1}(\hat{x}_1, \dots, \hat{x}_n)} \\
&= \left[\alpha - \beta \left(\frac{\alpha \hat{x}_1 - \hat{x}_2}{\beta \hat{x}_1} \right) \right] \left[\alpha - \beta \hat{x}_1 \left(\frac{\alpha \hat{x}_1 - \hat{x}_2}{\beta \hat{x}_1} \right) \right] + \left[-\beta \hat{x}_1 \right] \left[\gamma \hat{x}_1 + \delta \hat{x}_1 \left(\frac{\alpha \hat{x}_1 - \hat{x}_2}{\beta \hat{x}_1} \right) \right] \\
&= \frac{\hat{x}_2^2}{\hat{x}_1} + \delta \hat{x}_1 \hat{x}_2 - \gamma \hat{x}_2 - \alpha \delta \hat{x}_1^2 + \alpha \gamma \hat{x}_1
\end{aligned} \tag{4.15}$$

Summarizing the results of Eqs. 4.14-4.15 obtains the controller canonical realization of the Lotka-Volterra system:

$$\begin{aligned}
\dot{\hat{x}}_1 &= \hat{x}_2 \\
\dot{\hat{x}}_2 &= \frac{\hat{x}_2^2}{\hat{x}_1} + \delta \hat{x}_1 \hat{x}_2 - \gamma \hat{x}_2 - \alpha \delta \hat{x}_1^2 + \alpha \gamma \hat{x}_1
\end{aligned} \tag{4.16}$$

Following Corollary 4.1.5, the second-order ODE realization of a single variable for the Lotka-Volterra system is:

$$\ddot{x}_1 = \frac{\dot{x}_1^2}{x_1} + \delta \dot{x}_1 x_1 - \gamma \dot{x}_1 - \alpha \delta x_1^2 + \alpha \gamma x_1 \tag{4.17}$$

To reverse the process and obtain the standard realization of the Lotka-Volterra system from the canonical realization, the following transformation is used:

$$\begin{aligned}
\hat{x}_1 &= T_1^{-1}(\tilde{x}_1, \tilde{x}_2) = \tilde{x}_1 \\
\hat{x}_2 &= T_2^{-1}(\tilde{x}_1, \tilde{x}_2) = \alpha \tilde{x}_1 - \beta \tilde{x}_1 \tilde{x}_2
\end{aligned} \tag{4.18}$$

Following Eq. 4.10, the hidden variables and realization that corresponds to the transformation in Eq. 4.18 can be determined:

$$\begin{aligned}
\begin{bmatrix} \dot{\tilde{x}}_1 \\ \dot{\tilde{x}}_2 \end{bmatrix} &= \mathbf{J}_T^{-1}(\tilde{\mathbf{x}}) \cdot \begin{bmatrix} T_2(\tilde{x}_1, \tilde{x}_2) \\ \hat{f}(T_1(\tilde{x}_1, \tilde{x}_2), T_2(\tilde{x}_1, \tilde{x}_2)) \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ \alpha - \beta\tilde{x}_2 & -\beta\tilde{x}_1 \end{bmatrix}^{-1} \begin{bmatrix} \alpha\tilde{x}_1 - \beta\tilde{x}_1\tilde{x}_2 \\ \frac{(\alpha\tilde{x}_1 - \beta\tilde{x}_1\tilde{x}_2)^2}{\tilde{x}_1} + \delta\tilde{x}_1(\alpha\tilde{x}_1 - \beta\tilde{x}_1\tilde{x}_2) - \gamma(\alpha\tilde{x}_1 - \beta\tilde{x}_1\tilde{x}_2) - \alpha\delta\tilde{x}_1^2 + \alpha\gamma\tilde{x}_1 \end{bmatrix} \\
&= \frac{1}{-\beta\tilde{x}_1} \begin{bmatrix} -\beta\tilde{x}_1 & 0 \\ -\alpha + \beta\tilde{x}_2 & 1 \end{bmatrix} \begin{bmatrix} \alpha\tilde{x}_1 - \beta\tilde{x}_1\tilde{x}_2 \\ \alpha^2\tilde{x}_1 - 2\alpha\beta\tilde{x}_1\tilde{x}_2 + \beta^2\tilde{x}_1\tilde{x}_2^2 - \beta\delta\tilde{x}_1^2\tilde{x}_2 + \beta\gamma\tilde{x}_1\tilde{x}_2 \end{bmatrix} \\
&= \begin{bmatrix} \alpha\tilde{x}_1 - \beta\tilde{x}_1\tilde{x}_2 \\ -\gamma\tilde{x}_2 + \delta\tilde{x}_1\tilde{x}_2 \end{bmatrix} \tag{4.19}
\end{aligned}$$

which obtains the original realization of the Lotka-Volterra system as defined in Eq. 4.11.

Thus, the process of discovering hidden variables involves using one search algorithm to find the higher-order ODE realization of a single variable that models the measured data set (Eq. 4.17), and a second search algorithm to search candidate transformations (Eq. 4.18) and determine whether the corresponding transformation (Eq. 4.19) provides a parsimonious description of the system dynamics.

4.1.3 Transformations of nonlinear dynamical systems with multiple observed variables

This section expands the framework for the discovering hidden variables to nonlinear dynamical systems from time-series data to systems with multiple observed variables. It begins with a definition of nonlinear dynamical systems with multiple observed vari-

ables. The relationship between the hidden variables and the order of the observable ODEs is discussed, leading to the definition of nonlinear dynamical systems with single coupled hidden dynamics. Next, the corresponding controllable canonical realization for single coupled hidden dynamical systems is defined along with transformations to and from alternate realizations.

Definition 4.1.8. Nonlinear dynamical system with multiple observed variables. *Following the notation in Definition 4.1.1, nonlinear dynamical systems with multiple observed variables are systems that have more than one observed variable, or $1 < m < n$. Since the state variables can be rearranged arbitrarily, the observed variables are defined as the first m state variables without any loss of generality. A nonlinear dynamical system with multiple observed variables is defined as:*

$$\begin{aligned}
 \dot{x}_1 &= f_1(x_1, \dots, x_n) \\
 &\vdots \\
 \dot{x}_n &= f_n(x_1, \dots, x_n) \\
 y_1 &= x_1 \\
 &\vdots \\
 y_m &= x_m
 \end{aligned} \tag{4.20}$$

Alternatively, the state variables can be divided into their observed and hidden subsets, where $\mathbf{x}_o \in \mathbb{R}^m$ and $\mathbf{x}_h \in \mathbb{R}^{n-m}$ correspond to the observed and hidden variables, respectively, and $\mathbf{x} = [\mathbf{x}_o \ \mathbf{x}_h]^T \in \mathbb{R}^n$. Thus, Eq. 4.20 can be rewritten in vector form as:

$$\begin{aligned}
 \dot{\mathbf{x}}_o &= \mathbf{f}_o(\mathbf{x}_o, \mathbf{x}_h) \\
 \dot{\mathbf{x}}_h &= \mathbf{f}_h(\mathbf{x}_o, \mathbf{x}_h) \\
 \mathbf{y} &= \mathbf{x}_o
 \end{aligned} \tag{4.21}$$

Given a system with m observed variables, the remaining hidden variables can be transformed arbitrarily while maintaining an identical output of the system. An alternate

realization of the state space, $\tilde{\mathbf{x}}$, is defined through the nonlinear transformation:

$$\begin{aligned}
\tilde{x}_1 &= T_1(x_1, \dots, x_n) = x_1 \\
&\vdots \\
\tilde{x}_m &= T_m(x_1, \dots, x_n) = x_m \\
\tilde{x}_{m+1} &= T_{m+1}(x_1, \dots, x_n) \\
&\vdots \\
\tilde{x}_n &= T_n(x_1, \dots, x_n)
\end{aligned} \tag{4.22}$$

where the $T : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is nontrivial bijective map.

Definition 4.1.9. Observable ordinary differential equations. *Observable ordinary differential equations are ODEs that contain only the observed state variables and no hidden variables.*

Theorem 4.1.10. Existence of first-order observable ODEs. *For a dynamical system, a nontrivial, first-order, observable ODE exists if and only if the corresponding state transition function does not contain hidden variables.*

Proof. (Necessity) A state transition function is defined as a nontrivial, first-order ODE. If it is observable, then by definition, it cannot contain hidden variables.

(Sufficiency) If there are no hidden variables in the state transition function, then the state transition function, which is a first-order ODE, is observable. \square

Corollary 4.1.11. Existence of hidden variables for higher-order observable ODEs. *For a dynamical system, there exists only nontrivial, higher-order, observable ODEs if and only if the corresponding state transition function contains hidden variables.*

Proof. Corollary 4.1.11 is the logical negation of Theorem 4.1.10. \square

Theorem 4.1.10 and Corollary 4.1.11 are straightforward, but their implications are important enough that they are worth stating explicitly: the existence of hidden variables can be determined solely by the order of an observable ODE. Combined with Corollary 4.1.5, an n^{th} -order, observable ODE requires the existence of n hidden variables. Note, this result does not extend to a system of observable ODEs since the state transition functions may depend on the same hidden variables. For example, it is possible for a system with two observed variables and one hidden variables to produce two, linearly independent observable, second-order ODEs.

Nonetheless, Theorem 4.1.10 can still be leveraged to guide the discovery of hidden variables under specific circumstances. Experimentally, the process begins by measuring as many independent variables as possible. A search algorithm is then used to infer observable ODEs for each of the observed variables. If a first-order ODE is inferred, then there are no hidden variables in those dynamics. Likewise, if a higher-order ODE is inferred, then those dynamics contain hidden variables. If the system is a nonlinear dynamical system with one hidden coupled dynamics (Definition 4.1.12), then the combination of inferred ODEs can be used to discover hidden variables.

Definition 4.1.12. *Nonlinear dynamical system with one hidden coupled dynamics.* Dividing the observed state variables into uncoupled and coupled, where $\mathbf{x}_u \in \mathbb{R}^{m-1}$ and $\mathbf{x}_c \in \mathbb{R}$ correspond to the uncoupled and coupled observed variables, respectively, and $\mathbf{x}_o = [\mathbf{x}_u \ \mathbf{x}_c]^T \in \mathbb{R}^m$. A nonlinear dynamical system with one hidden coupled dynamics is defined as:

$$\begin{aligned}\dot{\mathbf{x}}_u &= \mathbf{f}_u(\mathbf{x}_u, \mathbf{x}_c) \\ \dot{\mathbf{x}}_c &= \mathbf{f}_c(\mathbf{x}_u, \mathbf{x}_c, \mathbf{x}_h) \\ \dot{\mathbf{x}}_h &= \mathbf{f}_h(\mathbf{x}_u, \mathbf{x}_c, \mathbf{x}_h) \\ \mathbf{y} &= [\mathbf{x}_u \ \mathbf{x}_c]^T\end{aligned}\tag{4.23}$$

The nonlinear dynamical system with one hidden coupled dynamics is named due to its property that all the interactions between the observed and hidden variables can only occur through the directly coupling of one state transition function. Note, this definition does not restrict the number of hidden variables, just the amount of coupling between the observed and hidden variables.

In this work, only systems of nonlinear dynamical systems with multiple observations that have only one hidden coupled dynamics will be considered. Although this limits the types of systems that can be analyzed, it is large subset of systems that includes many popular higher order systems such as the Lorenz attractor [117], the Rossler attractor [166] and the Chua circuit [127], which are capable of generating a wide variety of behaviors including chaos.

Definition 4.1.13. Modified controller canonical realization. *The modified controllable canonical realization, with a state space notation $\hat{\mathbf{x}} = [\hat{\mathbf{x}}_u \ \hat{\mathbf{x}}_k]^T \in \mathbb{R}^m$, where $\hat{\mathbf{x}}_u \in \mathbb{R}^{m-1}$ and $\hat{\mathbf{x}}_k \in \mathbb{R}^{n-m+1}$ are the uncoupled observed variables and canonical state variables, respectively, is defined as follows:*

$$\begin{aligned}
\dot{\hat{\mathbf{x}}}_u &= \hat{\mathbf{f}}_u(\hat{\mathbf{x}}_u, \hat{\mathbf{x}}_k) \\
(\dot{\hat{\mathbf{x}}}_k)_1 &= (\dot{\hat{\mathbf{x}}}_k)_2 \\
&\vdots \\
(\dot{\hat{\mathbf{x}}}_k)_{n-m} &= (\dot{\hat{\mathbf{x}}}_k)_{n-m+1} \\
\dot{\hat{\mathbf{x}}}_{n-m+1} &= \hat{\mathbf{f}}_k(\hat{\mathbf{x}}_u, \hat{\mathbf{x}}_k)
\end{aligned} \tag{4.24}$$

Corollary 4.1.14. Ordinary differential equation realization. *Every modified controllable canonical realization has an equivalent realization consisting of a set of coupled $m - 1$ first-order, observable ODEs and one $(n - m + 1)^{th}$ -order, observable ODE.*

Proof. The \hat{f}_u is a set of first-order set of coupled $m - 1$ first-order, observable ODEs and \hat{f}_c is one $(n - m + 1)^{\text{th}}$ -order, observable ODE, with the following change of notation:

$$(\hat{x}_k)_i = \frac{d^{i-1}x_c}{dt^{i-1}}. \quad \square$$

Theorem 4.1.15. *Transformation to the modified controllable canonical realization.*

If the following transformation, $\hat{\mathbf{x}} = \mathbf{T}(\mathbf{x})$, is bijective:

$$\begin{aligned} \dot{\hat{\mathbf{x}}}_u &= \mathbf{T}_u(\mathbf{x}_u, x_c, \mathbf{x}_h) = \mathbf{x}_u \\ (\dot{\hat{x}}_k)_1 &= \mathbf{T}_{k,1}(\mathbf{x}_u, x_c, \mathbf{x}_h) = x_c \\ (\dot{\hat{x}}_k)_2 &= \mathbf{T}_{k,2}(\mathbf{x}_u, x_c, \mathbf{x}_h) = \frac{dx_c}{dt} = f_m(\mathbf{x}_u, x_c, \mathbf{x}_h) \\ &\vdots \\ (\dot{\hat{x}}_k)_{n-m+1} &= \mathbf{T}_{k,n-m+1}(\mathbf{x}_u, x_c, \mathbf{x}_h) = \frac{d^{n-m}x_c}{dt^{n-m}} = \sum_{j=1}^n \frac{\partial T_{k,n-m}}{\partial x_j}(\mathbf{x}_u, x_c, \mathbf{x}_h) \cdot f_j(\mathbf{x}_u, x_c, \mathbf{x}_h) \Big|_{x_i = T_i^{-1}(\hat{x}_1, \dots, \hat{x}_n)} \end{aligned} \quad (4.25)$$

then there exists a modified controllable canonical realization of the system.

Proof. Note that \mathbf{T}_u is the trivial transformation and apply Theorem 4.1.6 to the \mathbf{T}_c transformation. \square

Theorem 4.1.16. *Transformation from the modified controllable canonical realization.* For a transformation $\hat{\mathbf{x}} = \mathbf{T}(\tilde{\mathbf{x}})$, if the Jacobian determinant of the transformation is non-zero, then the realization for the state space $\tilde{\mathbf{x}}$ can be determined from the controllable canonical realization.

Proof. First, modified controllable canonical realization and the transformation can be combined to define the following relationship:

$$\begin{aligned}
\dot{\hat{\mathbf{x}}}_u &= \mathbf{f}_u(\hat{\mathbf{x}}_u, \hat{\mathbf{x}}_k) = \mathbf{f}_u(\mathbf{T}_u(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_c, \tilde{\mathbf{x}}_h), \mathbf{T}_k(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_c, \tilde{\mathbf{x}}_h)) \\
(\dot{\hat{\mathbf{x}}}_k)_1 &= (\dot{\hat{\mathbf{x}}}_k)_2 = T_{k,1}(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_c, \tilde{\mathbf{x}}_h) \\
&\vdots \\
(\dot{\hat{\mathbf{x}}}_k)_{n-m} &= (\dot{\hat{\mathbf{x}}}_k)_{n-m-1} = T_{k,n-m-1}(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_c, \tilde{\mathbf{x}}_h) \\
(\dot{\hat{\mathbf{x}}}_k)_{n-m+1} &= \hat{\mathbf{f}}(\mathbf{T}_u(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_c, \tilde{\mathbf{x}}_h), \mathbf{T}_k(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_c, \tilde{\mathbf{x}}_h))
\end{aligned} \tag{4.26}$$

and the remainder follows Theorem 4.1.7. \square

The approach to discovering hidden variables with multiple observations from time series data is then defined by two search algorithms. The first algorithm infers the lowest order, observable ODE models for each observed variable that models the time series data. If the infer models are a set of coupled $m - 1$ first-order, observable ODEs and one $(n - m + 1)^{\text{th}}$ -order, observable ODE, then the model is converted to its corresponding controller canonical realization using Corollary 4.1.14. The second algorithm searches for parsimonious transformations by inferring candidate state space realizations from the controller canonical realization using Theorem 4.1.16.

4.1.4 Example realizations for the Chua circuit dynamical system

To illustrate the process of discovering hidden variables from a nonlinear dynamical system with a multiple observed variables, this section provides an example using the Chua circuit [127]. This system is a third-order, nonlinear dynamical system that was designed as a real-world of chaotic behavior. The circuit is designed with three passive components, two capacitors and one inductor, and one active nonlinear diode. The cubic

Chua diode will be analyzed instead of the traditional piecewise implementation due to its smooth and more consistent behavior [216]. The standard realization of the Chua circuit dynamics is:

$$\begin{aligned}\dot{x}_1 &= -\alpha x_1 + \beta x_2 - \gamma x_1^3 \\ \dot{x}_2 &= \delta x_1 - \delta x_2 + \epsilon x_3 \\ \dot{x}_3 &= -\kappa x_2 - \lambda x_3\end{aligned}\tag{4.27}$$

where x_1 and x_2 are the capacitor voltages, x_3 is an inductor current and $\alpha, \beta, \gamma, \delta, \epsilon, \kappa, \lambda > 0$ are all model parameters. For the sake of brevity and clarity, all the model parameters will be set to 1 for the remainder of this example, or $\alpha = \beta = \gamma = \delta = \epsilon = \kappa = \lambda = 1$.

The analysis begins by illustrating the difficulties in analyzing the Chua circuit using only a single variable. If only x_1 is observable, the corresponding higher-order ODE realization of the system is:

$$\ddot{x}_1 = -3\dot{x}_1 x_1^2 - 3\ddot{x}_1 - 6\dot{x}_1^2 x_1 - 6\dot{x}_1 x_2 - 3\dot{x}_1 - 2x_1^3 - 3x_1\tag{4.28}$$

Next, the transformation required to obtain the standard realization (Eq. 4.27) from the canonical realization is:

$$\begin{aligned}\hat{x}_1 &= \tilde{x}_1 \\ \hat{x}_2 &= -\tilde{x}_1 + \tilde{x}_2 - \tilde{x}_1^3 \\ \hat{x}_3 &= -4\tilde{x}_1^3 - 3\tilde{x}_1^5 - 3\tilde{x}_1^2 \tilde{x}_2 + \tilde{x}_3\end{aligned}\tag{4.29}$$

The complexity of Eq. 4.28 and Eq. 4.29 makes it difficult to infer using a search algorithm. In particular, searching for Eq. 4.28 numerically is challenging as there are a wide variety of expressions that approximate the data while searching for Eq. 4.29 is challenging due to the fact that set of equations must be found as a coupled solution. Although the Chua circuit with one observed variable could be inferred with superior

search algorithms, the system is significantly easier to infer if multiple variables could be measured.

First, if x_1 and x_2 are observed and x_3 is a hidden variable, the system satisfies criteria for a nonlinear dynamical system with one hidden coupled dynamics (Def. 4.1.12). The state transition function for x_1 only contains the observed variables, while the state transition function for x_2 is the only hidden coupled dynamics as it is the only ODE for an observed variable that contains hidden variables. The modified controller canonical form can be obtained using the transformation $\hat{\mathbf{x}} = \mathbf{T}(\mathbf{x})$:

$$\begin{aligned}\hat{x}_1 &= x_1 \\ \hat{x}_2 &= x_2 \\ \hat{x}_3 &= x_1 - x_2 + x_3\end{aligned}\tag{4.30}$$

And the corresponding inverse transformation, $\mathbf{x} = \mathbf{T}^{-1}(\hat{\mathbf{x}})$, is:

$$\begin{aligned}x_1 &= \hat{x}_1 \\ x_2 &= \hat{x}_2 \\ x_3 &= -\hat{x}_1 + \hat{x}_2 + \hat{x}_3\end{aligned}\tag{4.31}$$

Combining Eq. 4.30-4.31, Corollary 4.1.14 and Theorem 4.1.15, the corresponding set of observable ODEs for the system are:

$$\begin{aligned}\dot{x}_1 &= -x_1 + x_2 + x_1^3 \\ \ddot{x}_2 &= -\dot{x}_2 - x_2 - x_1^3\end{aligned}\tag{4.32}$$

The process of discovering hidden variables in systems with one hidden coupled dynamics involves using one search algorithm to find the set of observable ODEs that models the data for each observed variable (Eq. 4.32), and a second search algorithm to search candidate transformations (Eq. 4.30) and determine whether the corresponding transformation provides a parsimonious description of the system dynamics.

Thus, the search for hidden variables for the Chua circuit is significantly easier when two variables are observed. Comparing the observable ODEs in Eq. 4.32 with Eq. 4.28, it is clear that the complexity of the system of two ODEs is significantly less than the complexity of the single ODE. Furthermore, the two ODEs can be inferred independently, allowing the search to be divided into two smaller problems. Comparing the transformations in Eq. 4.30 with Eq. 4.29 again indicates that the complexity is significantly reduced and the Eq. 4.30 transformation is easier to find since there is only one nontrivial transformation as opposed to the pair of coupled transformations in Eq. 4.29.

4.2 Inferring ordinary differential equations of arbitrary order

Modeling dynamical systems is a critical component of many domains in science and engineering. Often represented as a set of differential equations that describe the relationships between the state variables and their rates of change. The form of these equations are usually derived by hand from first principles and the parameters of these equations are typically known in advance or identified through regression methods [116]. This forward modeling approach requires expert domain knowledge for each system of interest and, as a result, is limited by the experience and ingenuity of the expert [52, 98, 105, 107, 189]. Consequently, there is a growing interest in automated reverse-engineering approaches that are capable of generating the same dynamical models by merely analyzing observations of the system [211].

A major challenge in applying the reverse-engineering approach arises when the collected data does not include all of the state variables of the system. Since the goal of reverse engineering is to infer models with limited prior knowledge, the collected data does not necessarily include all of the appropriate state variables, or even the correct

number of variables. This issue is particularly troublesome in dynamical systems since the qualitative behavior, such as limit cycles or chaotic attractors, does not necessarily define the number of state variables [192]. Furthermore, even if there is sufficient theory to determine the state variables, obtaining the experimental measurements of these quantities may be difficult [162].

Although there are a variety of approaches that attempt to automate the model building process for dynamical systems, these approaches are limited in several aspects. Some methods restrict the inference to linear models [129] whereas others rely on numerical approximators, such as neural networks [82, 150]. While these methods may be capable of accurate predictions with sufficient complexity, their fixed-form parametric models do not shed any insights regarding the internal structure of the system. Recent work in automated reverse engineering of dynamical systems are capable of producing symbolic expressions, but they are still limited to data sets with complete information. Previous methods of inferring symbolic models of nonlinear dynamical systems that requires each state variable to be directly observable [17, 173]. However, approaches that rely on fully observed data require the user to define all of the state variables in advance.

This section presents a method for the automated synthesis of both the structure and parameters of an implicit ordinary differential equation from nonlinear dynamical systems given noisy time-series data with unobserved state variables. The method searches the space of symbolic expressions for a transformed representation using derivatives of the observed data.

For example, the Lotka-Volterra population dynamics tracks the number of lynxes and hares [209]; however, a mathematically identical system can be constructed using just the lynx population and its time derivative. The algorithm infers ordinary differential equations of arbitrary order, which effectively finds a governing equation of the

system using just the lynx population and its derivatives. To avoid trivial expressions, an approach is presented based on a principle of predictive accuracy, which leverages properties of differential equations to achieve a computationally efficient implementation. This method is demonstrated on eleven simulated and three physical dynamical systems spanning a variety of qualitative behaviors, from stable equilibria to chaotic attractors.

4.2.1 Model inference

This section introduces the two major contributions in the symbolic regression approach for the automated inference of nonlinear dynamical systems (Fig. 4.2A): the search for transformations of dynamical systems, which allows systems to be formally described despite the lack of observations for one or more state variables; and the predictability of higher-order derivatives, which discerns meaningful candidate expressions from trivial ones in a computationally efficient manner.

Transformations of dynamical systems

Since the observations of one or more state variables are absent from the data set, it is impossible to build a model of the system directly from observations that are included in the data set. However, the information from the missing state variables remains embedded in the observed data, as shown in Takens' theorem [197].

This property is best illustrated by considering transformations of the state space for dynamical systems. A n -dimensional dynamical system is specified by

$$\dot{\mathbf{x}} = \mathbf{v}(\mathbf{x}), \tag{4.33}$$

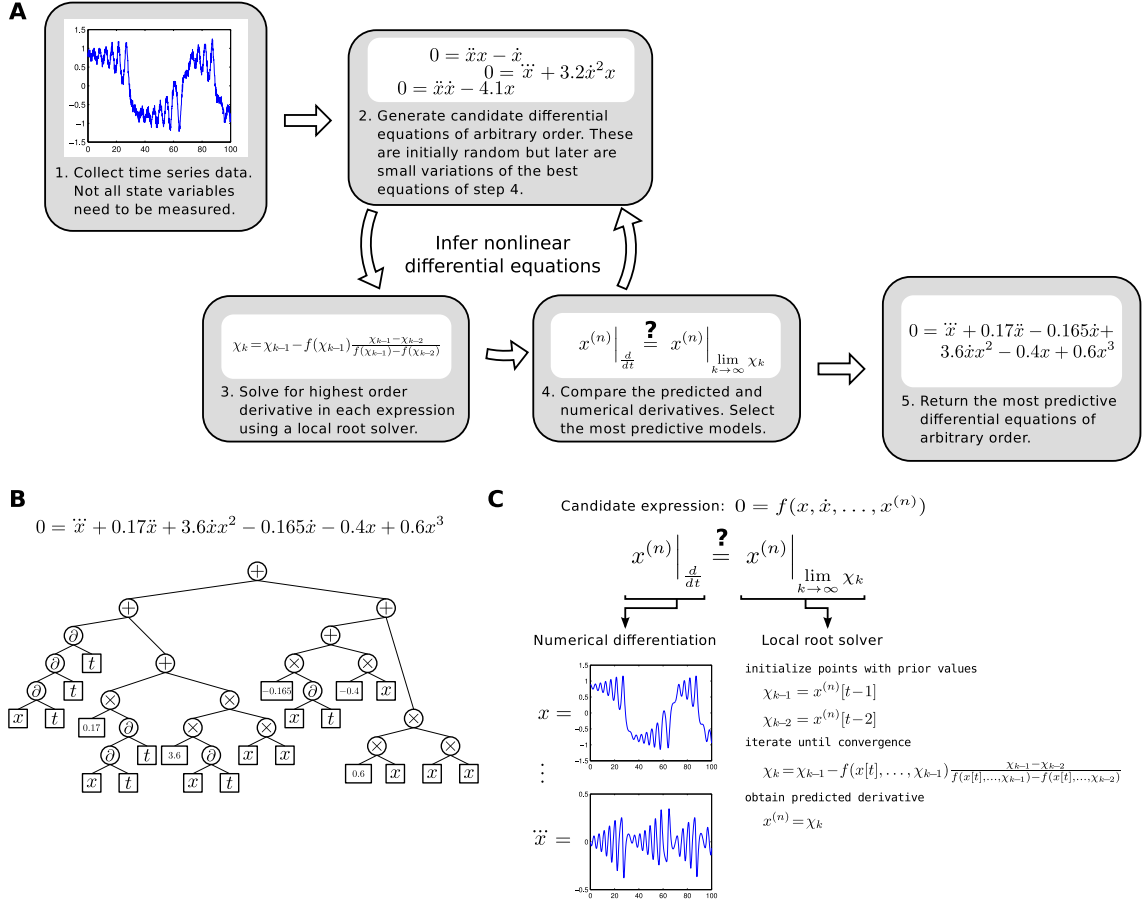


Figure 4.2: Process for discerning hidden dynamics. (A) A flowchart illustrating the iterative, symbolic regression process. The cycle (2,3,4) is repeated until a fixed period has elapsed or a predetermined objective error is achieved. (B) A binary tree representation for the third-order, chaotic Duffing oscillator. Implicit differential equations are represented using five operators: input variables, constants, addition, multiplication and differentiation. New candidate expressions are generated by modifying subtrees. (C) Computing an objective error based on the principle of predictability. The algorithm compares the highest order derivative of each expression, which is computed using two separate methods: numerical differentiation and a local root solver.

where $\mathbf{x} = [x_0, x_1, \dots, x_n]^T$ denotes a point in an n -dimensional state space $\mathbf{x} \in \mathbb{R}^n$, \mathbf{v} is the nonlinear vector field of the dynamical system and the overdot is the derivative with respect to time. Given an arbitrary bijective transformation, the state space of a dynamical system can be transformed to a topologically equivalent system [50]. For a wide class of systems, there exists a transformation that allows one to reformulate the standard representation, Eq. 4.33, as a single n -th order, autonomous, ordinary differential equation of a single variable

$$0 = f(x_0, \dot{x}_0, \dots, x_0^{(n)}), \quad (4.34)$$

where $x^{(n)}$ is the n -th derivative of x with respect to time. The conditions for valid transformations are described in detail in [50] for third-order systems, but can be readily extended to n -th order systems. For additional details, refer to Section 4.1.

The approach leverages this equivalence of dynamical systems to infer nonlinear models in the form of Eq. 4.34 without the loss of information. By including the order of the differential as part of the search process, the symbolic regression approach can represent arbitrary differential equations (Fig. 4.2B), which results in two advantageous properties. First, finding topologically equivalent state space representations only requires numerical differentiation of the observed time-series data. Second, the number of state variables necessary to model the system is automatically determined by the highest order derivative in the differential equation.

Predictability of higher order derivatives

The search for differential equations of arbitrary order naturally generates implicit expressions in the form of Eq. 4.34. A search using explicit expressions is unsuitable since the variables are not known in advance. However, searching for implicit expressions presents a significant challenge as there are infinite number of implicit expressions

that are trivially true yet describe no meaningful relationship in data [173]. For example, there are mathematical identities that are true for any data, such as $-3x + x(\dot{x} + 3) - x\dot{x} = 0$. Furthermore, there are expressions that are numerically true for any data when evaluated with arbitrary precision, such as $1/(100 + \dot{x}^2) \approx 0$. Thus, a more robust approach than simple substitution and evaluation is required to distinguish meaningful implicit expressions from poor ones [174].

The approach rewards implicit expressions by measuring their ability to predict relationships in the observed data (Fig. 4.2C). For each data point, the algorithm calculates a value for the highest order derivative in the candidate expression using two independent methods. The first method uses numerical differentiation of the time-series data [212]. The second method computes the highest order derivative using a local root finding method [155].

Each candidate expression defines a mathematical relationship between the derivatives of the state variables, and thus this approach substitutes computed values of the lower order derivatives and solves for the highest order derivative as an unknown. By leveraging the continuity of dynamical systems [20], the numerical method only needs to find the root that is nearest to the past values in the time-series data. In essence, this approach compares expression-based predictions of higher order derivatives with their numerical counterparts. This approach is able to identify trivial solutions due to their inability to make meaningful predictions.

4.2.2 Methods and algorithms

The section describes the tools, algorithms and metrics used in this study.

Symbolic regression

To find implicit ordinary differential equations, symbolic regression [104], an established search method based on evolutionary computation [60], is used. Unlike traditional regression approaches that begin with equations of a prespecified form and fit unknown model parameters, symbolic regression searches for both the parameters and equation structure simultaneously. The algorithm begins by randomly combining mathematical building blocks, such as algebraic operators, constants and observed variables, to form an initial set of expressions. New candidate expressions are generated by probabilistically recombining components of previous expressions and through random variations of subexpressions. The algorithm then evaluates how well each expression models the experimental data, and retains superior models for subsequent iterations while abandoning unpromising candidates. The algorithm terminates after a desired level of accuracy is achieved and returns the set of expressions that are most likely to correspond to the underlying mechanisms of the system. For additional details, refer to Section 1.3.2.

Symbolic expressions are represented as free-form lists of operations and parameters, as thus, both the form of the equation and its parameters are part of the search process. Expressions are represented as binary trees that consist of algebraic operations with numerical constants and symbolic variables at its leaves [128, 42]. The tree size was limited to the programs of 128 nodes as this is approximately the limit of human-interpretable equations or equations that could fit on a piece of paper. Ignoring the infinite parameter space, the search space is on the order of 10^{99} different parameterized equations.

Objective error

Due to the challenges of evaluating implicit differential equations, a significant amount of research was conducted on various approaches to this topic. Three different approaches were designed and they mostly revolved around using a combination of expression expansion, secant method and numerical integration.

Integration via finite differences. The initial approach for an objective error metric used numerical integration with a finite difference method. Each differential operator was expanded using its finite difference representation, references to previous time steps were substituted, and the current data point was solved using a local root finding algorithm. Since the approach used numerical integration, each data point was used as an initial condition and the algorithm integrated until a user defined error was reached. The fitness was the sum count of the integration steps as well as the residual error. For implementation details, refer to Algorithm 4.1.

This approach suffered from a number of numerical issues. First, this approach was inherently an Euler step integration where the order of the integrator was the order of the finite difference. Although higher-order finite differences could be used to reduce the integration error, this resulted in two fundamental tradeoffs. First, increasing the order of the finite difference resulted in a significant increase in computational effort. Second, finite difference methods are not robust to noisy data. Higher order difference methods inherently assume noiseless data and actually amplify the noise contributions for higher-order finite differences.

The second issue is that this approach had a root finding nonlinearity that was proportional to the nonlinearity of the expression and not the nonlinearity of the differential.

Algorithm 4.1: Integration via finite differences

```
1  input  → expression -  $f(x, \dot{x}, \dots, x^{(n)})$ 
2  output → fitness value = [count overflow]T
3
4  function eval_expression(expr, input_value, data, data_index) :
5      fd_expr = replace_derivative_with_finite_difference(expr)
6      numerical_expr = substitute_fd_using_data(data, data_index)
7      return resolve_expression(numerical_expr, input_value)
8
9
10 function secant_method(expr, data, data_index) :
11     pred_2 = data[data_index-2]
12     pred_1 = data[data_index-1]
13     pred    = 0
14
15     for i in range(secant_method_limit) :
16         expr_value_2 = eval_expression(expr, pred_2, data, data_index)
17         expr_value_1 = eval_expression(expr, pred_1, data, data_index)
18         if (expr_value_1 = expr_value_2) :
19             break
20         else :
21             pred = pred_1 - expr_value_1*
22                 (pred_1 - pred_2)/(expr_value_1 - expr_value_2)
23             pred_2 = pred_1
24             pred_1 = pred
25     return pred
26
27
28 fitness.count = 0
29 fitness.overflow = 0
30
31 stdev = get_highest_order(expr)
32 for i in range(data) :
33     error = 0
34     count = 0
35     while (error < user_limit) :
36         prediction = secant_method(expr, data, i)
37         error += abs(data[i] - prediction)
38         count += 1
39
40     fitness.count += count
41     fitness.overflow += fitness.error/stdev/range(data)
42
43 return fitness
```

For example, given the expression $0 = \ddot{x} + \dot{x}x + x$, the finite difference approach results in a root finding nonlinearity of x^2 due to the $\dot{x}x$ term, even though the highest differential term, \ddot{x} , is linear. This creates a significant problem since the accuracy of local root finders decreases significantly when the number of roots increases, and better initial conditions are required. As a result of these numerical issues, this approach was abandoned.

Integration via black-box, adaptive Runge-Kutta methods. The second approach used a Runge-Kutta method with adaptive stepsize for numerical integration [28]. The Runge-Kutta method required an explicit formulation of the expression, which was achieved through a local root finding algorithm. The state space of the system was simply derivatives of the time-series data and the initial conditions were estimates of these derivatives. By substituting the estimated state, the evolution of the system was obtained by solving for the highest order derivative using a local root finding algorithm.

The Runge-Kutta-Fehlberg method was applied [75], which uses an $O(n^4)$ method for computation and an $O(n^5)$ method for error computation, and is the basis for a variety of numerical integration methods including MATLAB's ode45. Effectively, by using the local root finder, the standard numerical integrator was able to be applied to an arbitrary black-box expression. For implementation details, refer to Algorithm 4.2.

Although this approach resolved many of the numerical method issues in the finite difference approach, it was computationally slow as it required many evaluations per time step. In the best-case analysis, the finite difference approach required $O(3n)$ function evaluations per data point, while the Runge-Kutta approach required $O(21n)$ function evaluations per data point. Furthermore, Runge-Kutta approach was a poor approach for predicting chaotic systems due to the sensitivity to initial conditions. However, it should be noted that this method was relatively robust to noise due to the fact it was comparing numerical integrations, which only depend on initial conditions.

Algorithm 4.2: Integration via black-box, adaptive Runge-Kutta methods

```
1 input → expression =  $f(x, \dot{x}, \dots, x^{(n)})$ 
2 output → fitness value = [count overflow]T
3
4 function eval_expression(expr, time, state, input_value) :
5     return sub_state_for_deriv_in_expr(expr, time, state, input_value)
6
7 function secant_method(expr, time, state, data) :
8     pred_2 = data[data_index-2]
9     pred_1 = data[data_index-1]
10    pred = 0
11    for i in range(secant_method_limit) :
12        expr_value_2 = eval_expression(expr, time, state, pred_2, data)
13        expr_value_1 = eval_expression(expr, time, state, pred_1, data)
14        if (expr_value_1 = expr_value_2) :
15            break
16        else :
17            pred = pred_1 - expr_value_1*
18                (pred_1 - pred_2)/(expr_value_1 - expr_value_2)
19            pred_2 = pred_1
20            pred_1 = pred
21    return pred
22
23 function rk45(expr, time, state, data, data_index, stepsize) :
24     target_stepsize = stepsize
25     while (error < error_limit) :
26         for i in range(5) :
27             dtime = time
28             dstate = state
29             for j in range(i) :
30                 dtime += rk_t[j]*target_stepsize
31                 dstate += rk_x[j]*estimate[j]
32             estimate[i] = secant_method(expr, dtime, dstate, data)
33             error = calculate_rk_error(estimate[i])
34             target_stepsize = target_stepsize/2
35     return calculate_pred(estimate[i])
36
37 fitness.count = 0
38 fitness.overflow = 0
39 stdev = get_highest_order(expr)
40 for i in range(data) :
41     error = 0
42     count = 0
43     state = get_state_via_finite_differences(data, i)
44     while (error < user_limit) :
45         prediction = rk45(expr, time[i], state, data, i, stepsize)
46         error += abs(data[i] - prediction)
47         count += 1
48     fitness.count += count
49     fitness.overflow += fitness.error/stdev/range(data)
50 return fitness
```

Comparing predicted derivatives. For the final approach, an objective error metric was used that compared the values of the highest order differential using two numerical approaches. Effectively, this approach converts the implicit equation into an explicit equation for the highest order differential using numerical methods. The first method for calculating the highest order differential determines the value by numerical differentiation methods, such as local polynomial regression; an n^{th} order differential is represented as $x^{(n)}\big|_{\frac{d}{dt}}$. The second method for calculating the highest order differential substitutes the numerical derivatives of the lower order differentials into the candidate expression and solves for the highest order differential using a local root finding method; an n^{th} order differential is represented as $x^{(n)}\big|_{\lim_{k \rightarrow \infty} \chi_k}$. A local root finding method is suitable since the data consists of measurements from a solution to a differential equation, and consequently, is Lipschitz continuous. Thus, values of the previous time steps are effective seeds for local root solvers.

Although there are many metrics for computing the error such as squared error, the objective error is defined as the normalized, mean absolute error:

$$E = \frac{1}{T} \sum_{t=0}^T \left| \frac{x^{(n)}[t]\big|_{\frac{d}{dt}} - x^{(n)}[t]\big|_{\lim_{k \rightarrow \infty} \chi_k}}{\sigma_{x^{(n)}}} \right| \quad (4.35)$$

where E is the objective error, T is the number of data points, χ_k is the k^{th} iteration of the local root solver and $\sigma_{x^{(n)}}$ is the standard deviation of the n^{th} derivative. It is critical to normalize the error since the derivatives can have significantly varying magnitudes. Normalizing by the standard deviation ensures that, regardless of the choice of derivative, an expression that does statistically no better than estimating the mean achieves an error of 1. The absolute error was chosen since it limits the emphasis of outliers. For implementation details, refer to Algorithm 4.3. This approach is illustrated for a damped harmonic oscillator in Fig. 4.3.

Algorithm 4.3: Comparing predicted derivatives

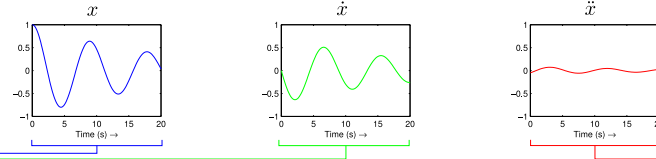
```
1 input  → expression -  $f(x, \dot{x}, \dots, x^{(n)})$ 
2 output → fitness value
3
4
5 function eval_expression(expr, input_value, numerical_deriv) :
6     return sub_num_deriv_in_expr(expr, input_value, numerical_deriv)
7
8
9 function secant_method(expr, data, numerical_deriv) :
10     pred_2 = data[data_index-2]
11     pred_1 = data[data_index-1]
12     pred   = 0
13     for i in range(secant_method_limit) :
14         expr_value_2 = eval_expression(expr, data, numerical_deriv)
15         expr_value_1 = eval_expression(expr, data, numerical_deriv)
16         if (expr_value_1 == expr_value_2) :
17             break
18         else :
19             pred = pred_1 - expr_value_1*
20                 (pred_1 - pred_2)/(expr_value_1 - expr_value_2)
21             pred_2 = pred_1
22             pred_1 = pred
23     return pred
24
25
26 numerical_deriv = get_numerical_derivatives(data)
27 stdev = get_highest_order(expr)
28 for i in range(data) :
29     prediction = secant_method(expr, data, numerical_deriv)
30     error += abs(numerical_deriv[highest_order] - prediction)
31 fitness = error/stdev/range(data)
32 return fitness
```

There are several important facts to note about this approach. First, Algorithm 4.3 is significantly much simpler Algorithm 4.1 and Algorithm 4.2. This simplicity is also reflected in the computational effort and is much faster to compute than the other methods. Next, unlike the other two approaches, Algorithm 4.3 relies on comparing derivatives as opposed to integrals. The tradeoff in this regards is that it is comparing local information in the form of state-space vector flows, as opposed to state trajectories. Inherently, local information is more susceptible to noise but is easier to compute and allows for the comparison of chaotic systems.

A. Candidate differential equation model

$$0 = \ddot{x} + 0.1\dot{x} + 0.5x$$

B. Numerically differentiate the time series data



C. Calculate predicted highest order time derivative via local root finding method

C-1. Setup for secant method

$$\ddot{x}_m[t] = \ddot{x}_{m-1}[t] - f(\ddot{x}_{m-1}[t]) \frac{\ddot{x}_{m-1}[t] - \ddot{x}_{m-2}[t]}{f(\ddot{x}_{m-1}[t]) - f(\ddot{x}_{m-2}[t])}$$

$$\rightarrow \ddot{x}_m[t] = \ddot{x}_{m-1}[t] - (\ddot{x}_{m-1}[t] + 0.1\dot{x}[t] + 0.5x[t]) \frac{\ddot{x}_{m-1}[t] - \ddot{x}_{m-2}[t]}{(\ddot{x}_{m-1}[t] + 0.1\dot{x}[t] + 0.5x[t]) - (\ddot{x}_{m-2}[t] + 0.1\dot{x}[t] + 0.5x[t])}$$

$$\rightarrow \ddot{x}_m[t] = \ddot{x}_{m-1}[t] - (\ddot{x}_{m-1}[t] + 0.1\dot{x}[t] + 0.5x[t]) \frac{\ddot{x}_{m-1}[t] - \ddot{x}_{m-2}[t]}{(\ddot{x}_{m-1}[t] + 0.1\dot{x}[t] + 0.5x[t]) - (\ddot{x}_{m-2}[t] + 0.1\dot{x}[t] + 0.5x[t])}$$

C-2. Iterate secant method

$$\ddot{x}_{m-1}[t] = \ddot{x}[t-1]$$

$$\ddot{x}_{m-2}[t] = \ddot{x}[t-2]$$

for ($m = 2; m < m_{\text{limit}}; m = m + 1$) :

$$\ddot{x}_m[t] = \ddot{x}_{m-1}[t] - (\ddot{x}_{m-1}[t] + 0.1\dot{x}[t] + 0.5x[t]) \frac{\ddot{x}_{m-1}[t] - \ddot{x}_{m-2}[t]}{(\ddot{x}_{m-1}[t] + 0.1\dot{x}[t] + 0.5x[t]) - (\ddot{x}_{m-2}[t] + 0.1\dot{x}[t] + 0.5x[t])}$$

if ($|f(\ddot{x}_{m-1}[t]) - f(\ddot{x}_{m-2}[t])| < \epsilon$) :

return $\ddot{x}_m[t]$

else :

$$\ddot{x}_{m-2}[t] = \ddot{x}_{m-1}[t]$$

$$\ddot{x}_{m-1}[t] = \ddot{x}_m[t]$$

D. Compare predicted derivative with numerical derivative

$$f_{ODE} = -\frac{1}{N} \sum_{t=2}^N \left| \ddot{x}[t] \Big|_{\lim_{m \rightarrow \infty} \ddot{x}_m} - \ddot{x}[t] \frac{d}{dt} \right|$$

Figure 4.3: An example of calculating the fitness of a differential equation model for a damped harmonic oscillator. Beginning with the candidate model (A), the time series data is differentiated numerically (B). The highest order time derivative is calculated using a local root finding method (C) and compared with the numerical derivative for the fitness metric (D).

Numerical differentiation and smoothing

Numerical differentiation and smoothing of the time-series data was achieved using a local polynomial regression method [212]. To obtain the n^{th} order derivative, a polynomial of the $n + 2$ degree was fit to a bin of the k^{th} nearest points in the time domain. The bin size was determined by 10-fold cross-validation.

Local root finder

The local root finding method was implemented with the Secant method [155]. This numerical approach was chosen due to its ease of implementation. Approaches such as Newton's method [155] require symbolic differentiation, while approaches such as the Bisection method [155] requires that the initialization points are on either side of the root, which is a property that cannot be easily guaranteed for arbitrary expressions.

Implementation details

The algorithm used an Age-Fitness Pareto selection method [175] with rank predictors [176] and variance trainers [119]. The operator probabilities are 2.5% and 70% for mutation and crossover, respectively. The encoding is a binary tree with a maximum of 128 nodes. The operation set contained addition, multiplication and differentiation with the addition of sine, cosine and tangent functions for the pendulum system. Furthermore, a hill-climber was used for parameter optimization [169]. The algorithm was executed on a single core on an Intel 2.8GHz processor with a population of 256 candidate expressions. The fitness predictor population contained 8 predictors, each with 8 indices to any point in the training data set that has sufficient neighbors for numerical differentiation. Predictors are evolved using deterministic crowding [124], with 12.5% mutation and 50% crossover. The trainer population contained 8 trainers, which were selected every 100 generations. The algorithm was executed for 2.5×10^6 generations, which resulted in approximately 5×10^{11} evaluations.

Equation complexity

Measuring equation complexity required all equations to be expressed in a common format. A sum-of-product representation was chosen due to ease of interpretation and implementation, which was achieved through algebraic expansion. When counting the tree size, the size could differ by 2 depending on whether the implicit equation had a common scalar factor. For example, $0 = 2\dot{x} + 6x$ has a size of 9 although it could be further simplified to $0 = \dot{x} + 3x$, which has a size of 7.

Noise synthesis

Additive Gaussian noise was introduced to the synthetic data sets at various levels. The signal-to-noise ratio was defined as follows:

$$S = \frac{\sigma_n}{\sigma_D} \quad (4.36)$$

where S is the signal-to-noise ratio, σ_n is the standard deviation of the Gaussian distribution and σ_D is the standard deviation of state variable. The noise was added after generating the data set.

4.2.3 Results

This algorithm was applied to eleven synthetic systems and three physical systems.

Application to synthetic systems

The method was applied to eleven systems (Table 4.1), which is further organized into three sets. The first set of systems are variations of the Duffing oscillator, a classical

nonlinear system [49, 72]. The set includes the standard second order system, a third order system driven with a low pass feedback [114], as well as a fourth order system driven with a sinusoidal input. The second set are variations of the Van der Pol oscillator, another classical nonlinear system [203, 204], and includes the standard second order system, a third order system driven with a low pass feedback, as well as a fourth order system driven with a sinusoidal input. The third set consists of Sprott’s simple chaotic systems [190], all of which are third-order systems. These set of systems span a wide range of qualitative behaviors, including stable equilibrium, limit cycles and chaotic attractors.

The data for these systems were generated using MATLAB’s `ode45` numerical solver. The data set for the second and third order Duffing and Van der Pol oscillators, as well as the Sprott equations, contained only data from the time variable and a single state variable, denoted as x in Table 4.1. A second data set for the third and fourth order Duffing and Van der Pol oscillators, contained data of the time variable as well as two state variables, denoted as x and y in Table 4.1.

Note that the collected data sets are missing at least one state variable for each system as the order of the differential equation is larger than the number of variables. Furthermore, measurement noise was simulated by adding Gaussian white noise to each state variable with a signal-to-noise ratio of 0%, 1%, and 10%. Plots of all eleven noiseless data sets and plots of the noise corruption for the third order Duffing oscillator are shown in Fig. 4.4 and Fig. 4.5, respectively.

Table 4.1 reports the successful inference of all eleven systems without any domain knowledge. For each system, eight independent trials were performed. In all of the experiments, the systems were inferred using the same algorithm, with perfect reproducibility given sufficient computational effort.

Table 4.1: Inference of noisy, synthetic systems

Synthetic system	Description
Duffing oscillators	
1. $0 = \ddot{x} + 0.25\dot{x} - 0.8x + 1.2x^3$	Unforced; stable equilibrium
2. $0 = \ddot{x} + 0.17\ddot{x} - 0.165\dot{x} + 3.6\dot{x}x^2 - 0.4x + 0.6x^3$	Forced with low-pass feedback; chaotic attractor
3. $0 = \ddot{x} - 0.33\dot{x} - 0.8x + 1.2x^3 + 1.6y$ $0 = \dot{y} + 0.5y - 0.5\dot{x}$	Forced with low-pass feedback; chaotic attractor
4. $0 = \ddot{x} + 0.25\dot{x} - 0.8x + 1.2x^3 + 0.4y$ $0 = \ddot{y} + 1.1y$	Forced with sinusoidal input; chaotic attractor
Van der Pol oscillators	
5. $0 = \ddot{x} + 2\dot{x} + 2\dot{x}x^2 + x$	Unforced; limit cycle
6. $0 = \ddot{x} + 0.5\ddot{x} + 2\ddot{x}x^2 + 4\dot{x}^2x + 5\dot{x}x^2 + 0.25\dot{x} + 2.5x$	Forced with low-pass feedback; limit cycle
7. $0 = \ddot{x} + 2\dot{x} + 2\dot{x}x^2 + x + 1.7y$ $0 = \dot{y} + 2.5y - 2.5\dot{x}$	Forced with low-pass feedback; limit cycle
8. $0 = \ddot{x} + 2\dot{x} + 2\dot{x}x^2 + x - 1.8y$ $0 = \ddot{y} + 1.7y$	Forced with sinusoidal input; chaotic attractor
Sprott equations	
9. $0 = \ddot{x} + 2.8\dot{x} - x - x^2$	Unforced; chaotic attractor
10. $0 = \ddot{x} + 0.5\ddot{x} + \dot{x} + x + x^2$	Unforced; chaotic attractor
11. $0 = \ddot{x} + 0.7\ddot{x} + \dot{x} - x + x^3$	Unforced; chaotic attractor

Eight independent trials were conducted for each of the eleven synthetic systems shown above. The best model is shown, although each trial found the same expression with a $< 1\%$ error for parameter values. The observed state variables are denoted as x and y , respectively. Systems 2 and 3, as well as 6 and 7, are mathematically equivalent and only differ in the number of observed variables.

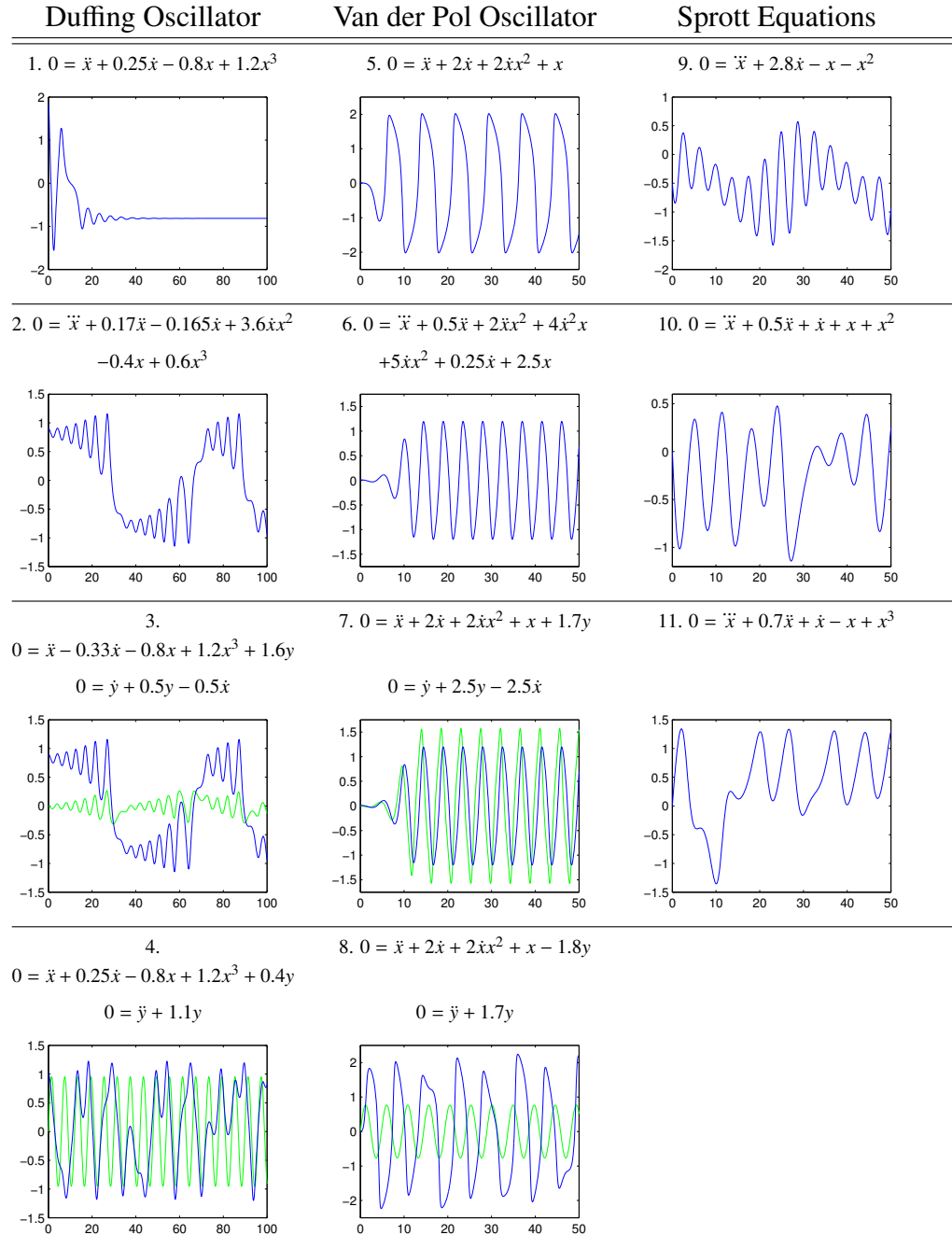


Figure 4.4: Plots of the noiseless data sets for all eleven synthetic system along with their respective differential equations. The state variable x is shown in blue and the state variable y is shown in green.

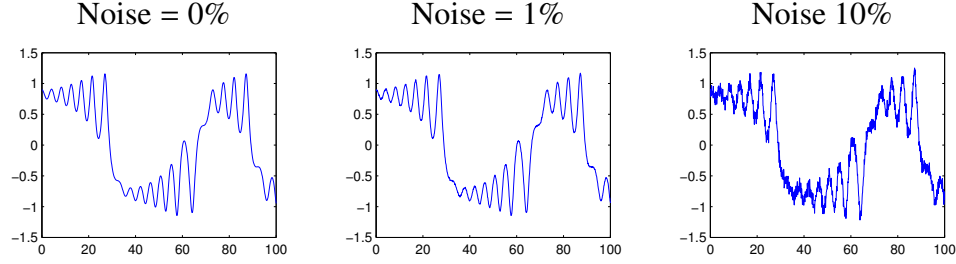


Figure 4.5: Plots of the third-order, chaotic Duffing oscillator with various levels of noise corruption.

Fig. 4.6A shows the mean objective error compared to the computational effort for the third order Duffing oscillator. For this system, numerically accurate models are achieved with approximately 10^{10} expression evaluations. Since the algorithm relies on heuristic methods, it is impossible to predict in advance the computational effort required to find the implicit differential equation model. However, there is a weak correlation between the complexity of the expression and the required computational effort across different systems, and thus, similar systems can be used as a reference to predict the computational effort. The convergence plots for all eleven systems are shown in Fig. 4.7.

A significant challenge in data-driven model fitting with finite data sets is dealing with overfitting, which occurs when statistical models describes random noise as opposed to the underlying relationship. Overfitting occurs in symbolic models when they achieve a lower objective error by finding expressions with excessive number of terms, while underfitting is the complementary phenomenon of finding simple but inaccurate models [42]. Thus, there is a tradeoff between the predictive accuracy and expression complexity that cannot be specified in advance for a given system [173]. Rather than producing a single result, the algorithm presents a small set of candidate expressions that are Pareto optimal with regards to the objectives of accuracy and complexity, where accuracy is the objective function and complexity is the size of the corresponding tree

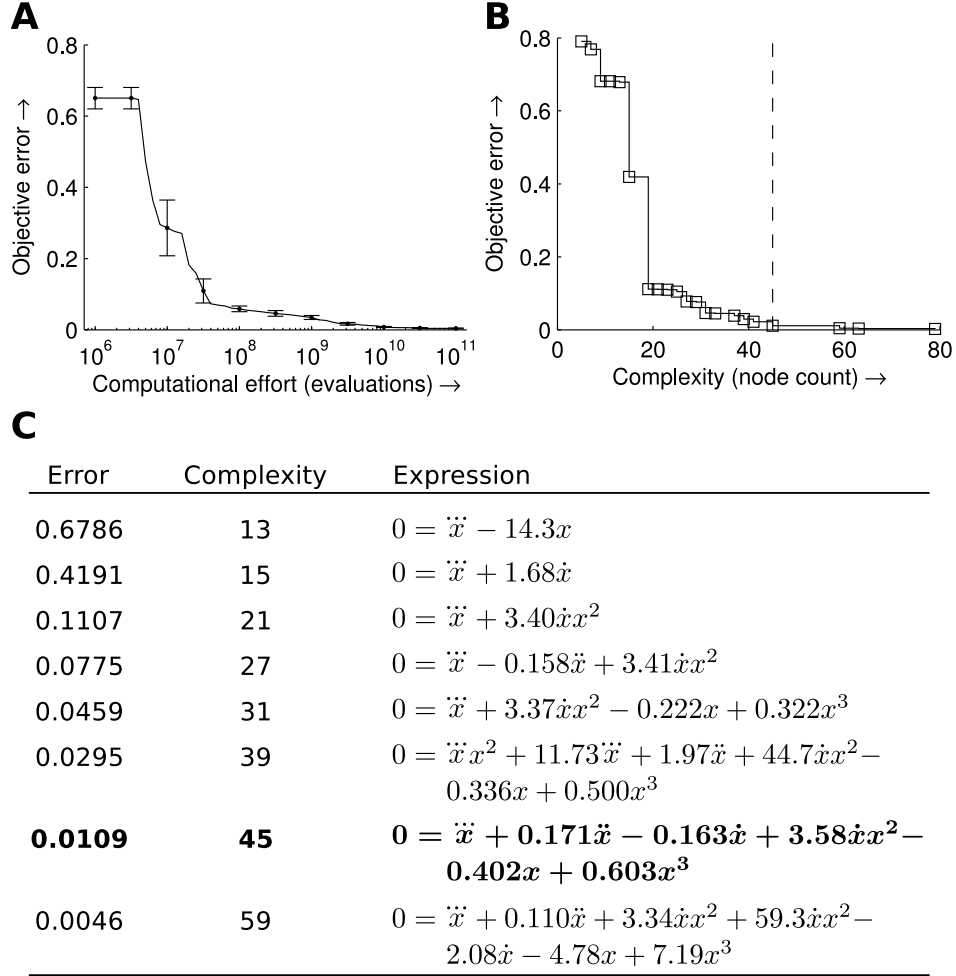


Figure 4.6: Accuracy-complexity tradeoff. (A) A plot of the objective error as a function of the computational effort for the third-order, chaotic Duffing oscillator. Error bars indicate standard error ($n = 8$). (B) A plot of the final accuracy-complexity Pareto optimal set for the third-order, chaotic Duffing oscillator. The complexity of true expression for the system is indicated by the dashed line. (C) Selected expressions from the Pareto optimal set in B. The true expression for the system is indicated in boldface.

expression. Every expression in this Pareto set represents a model that is non-dominated in either objective. For additional details, refer to Section 1.3.4.

The Pareto set for the third order Duffing oscillator is shown in Fig. 4.6B. The Pareto set tends to contain a sharp transition where the predictive ability jumps rapidly at a spe-

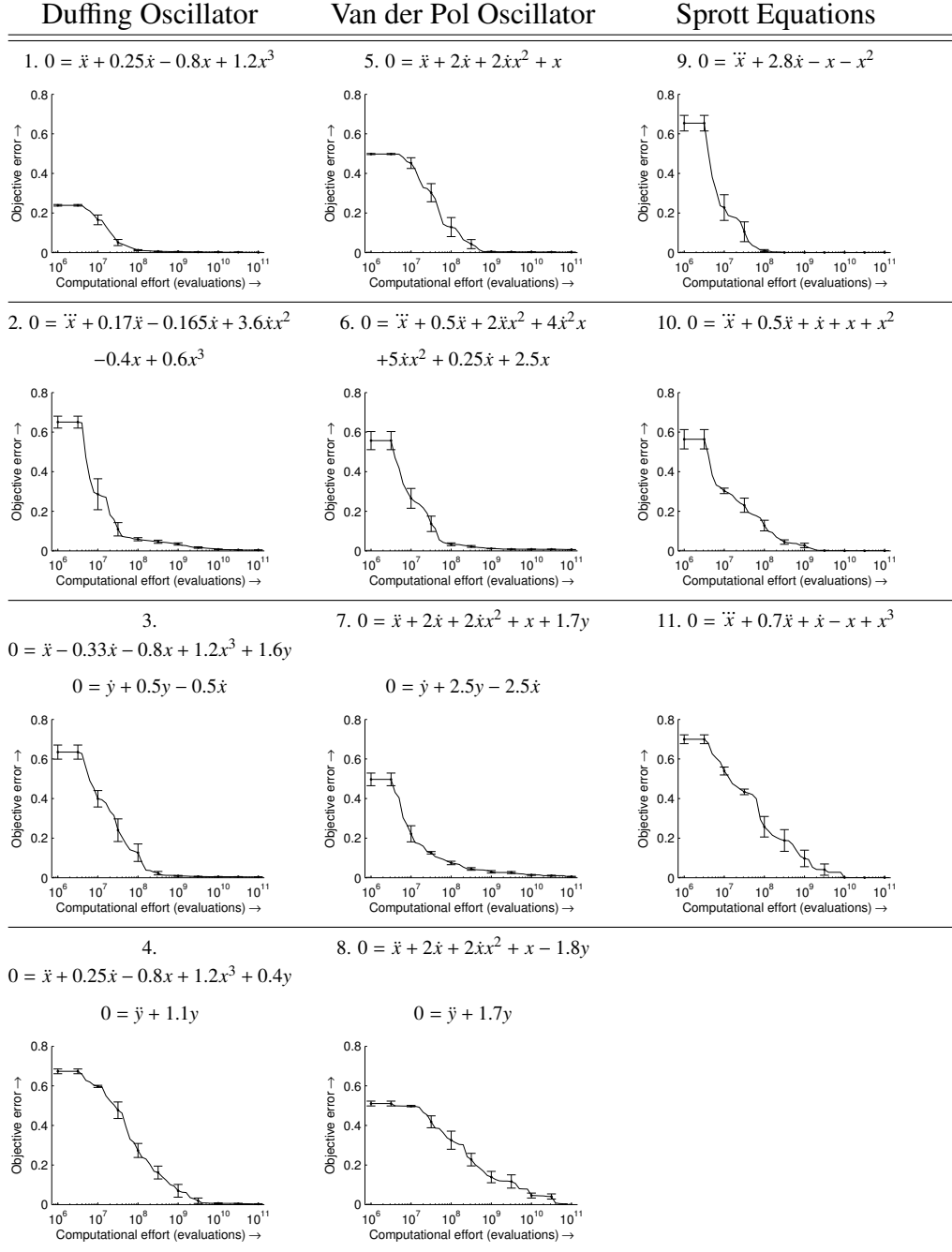


Figure 4.7: Plots of the objective error as a function of computational effort for all non-trivial expressions for the synthetic systems. Error bars indicate standard error ($n=8$).

cific complexity. Overfit expressions require significantly greater complexity to achieve only marginal improvements in predictive accuracy. In all of the experiments, this transition highlights the true expression of the dynamical system. The Pareto sets for all eleven systems are shown in Fig. 4.8.

There are several factors that affect the scalability of the algorithm. The primary factor is the complexity of the true equations. In the worst case, the computational effort required scales exponentially with the size of the tree expression; although in the experiments, the algorithm performed significantly better than the worst case analysis. This factor can be partially mitigated computationally through parallelization as the algorithm is readily parallelizable and many candidate expressions need to be evaluated simultaneously.

A related factor is the number of missing state variables, which indirectly affects the complexity of the system. As the number of unobserved state variables increases, higher order derivatives are required to generate the corresponding differential equation model. For nonlinear systems, differentiation requires the chain rule, which inherently increases the complexity of the expression [191]. Thus, increasing the number of observed state variables significantly reduces the required computational effort. For example, with the third order Duffing oscillator, successful inference with only a single state variable requires almost a ten-fold increase in computational effort over the exact same system with two state variables (Fig. 4.9). The effect of the number of missing variables is also evident in the Pareto set (Fig. 4.8): the true expression has a lower complexity and is easier to identify for the two state variable systems compared to their single state variable counterparts. Furthermore, the algorithm is unable to find the the differential model for the fourth order Duffing and Van der Pol oscillators with a single variable, due to the technical limitations of obtaining accurate numerical derivatives of time-series data.

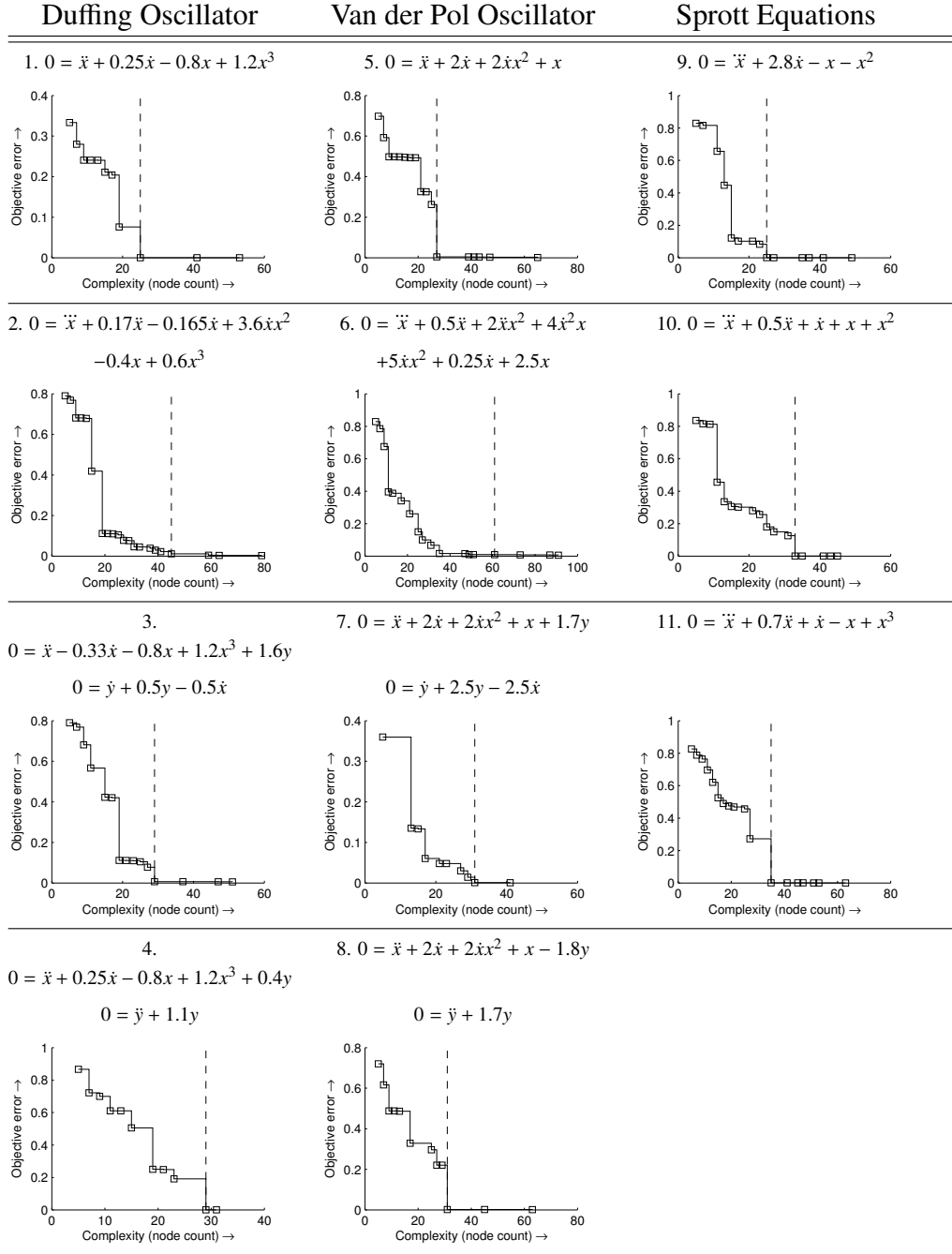


Figure 4.8: Plots of the final accuracy-complexity Pareto optimal set for all non-trivial expressions for the synthetic systems. The complexity of true expression for the system is indicated by the dashed line.

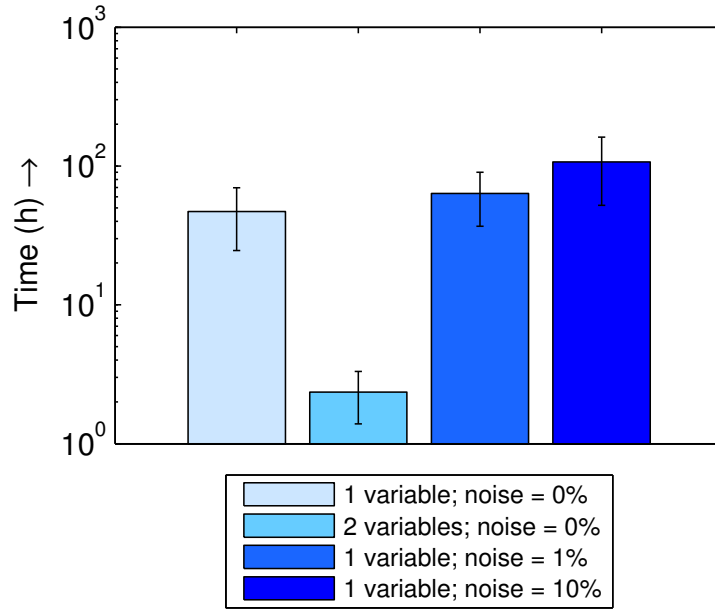


Figure 4.9: The computational time required to infer the third-order chaotic Duffing oscillator. Error bars indicate standard deviation ($n = 8$)

However, the algorithm succeeds for the same fourth order system when it is provided with two state variables.

Another factor that affects the scalability of the algorithm is the amount of noise (Fig. 4.9). The algorithm depends on obtaining accurate numerical derivatives and high levels of measurement noise makes this computation difficult. Furthermore, this effect is magnified as the number of missing state variables increases, since this necessitates higher order differentials. Nonetheless, the algorithm was still successful even in the presence of 10% measurement noise at the cost of greater computational effort.

Note that all of the data in these experiments is captured from a single trajectory of the dynamical system. Data sets that include additional trajectories with different initial conditions may yield a more complete description of the phase space and result

in more robust results with faster model inference. In this regards, chaotic attractors are preferable to periodic behavior due to the wealth of information about the state space. In fact, the Sprott equations were relatively easy to infer due to their simple expressions with low tree size and chaotic behavior. In comparison, the third order Van der Pol oscillator with a single variable was surprisingly difficult. The expression was the largest in the set of synthetic systems and the dynamics collapsed to a limit cycle on a two dimensional manifold. Thus, the data to required to distinguish the third order dynamics was only evident in the transient behavior, which was relatively limited in the data set.

Application to physical systems

The algorithm was also applied to three physical systems typically found in undergraduate physics education: a two-spring single-mass air-track oscillator, a three-spring double-mass air-track oscillator, and a pendulum (Table 4.2). The data was obtained from a previously published data set [173], and motion-tracking software was used to record the positions of the systems with respect to time. For each system, eight independent trials were performed. A minor change was applied to the algorithm for the pendulum data set: trigonometric functions were supplied in addition to the standard algebraic operators.

For the single-mass oscillator, the algorithm successfully inferred the equation of motion for the system when provided with just the position of the mass. The effect of drag was within the margins of noise and, consequently, was not modeled by the algorithm. Without any prior knowledge of physics, it automatically determined that the system was best modeled with a second order differential equation using the form of a harmonic oscillator.

Table 4.2: Inference of physical systems

System description	Inferred expression
1. Double-spring single-mass oscillator	$0 = \ddot{x} + 6.21x$
2. Triple-spring double-mass oscillator	$0 = \ddot{x} + 78.3x - 70.7y$ $0 = \ddot{y} + 70.7y - 78.3x$
3. Pendulum	$0 = \ddot{\theta} + 1.31 \sin(\theta)$

Eight independent trials were conducted for each of the three physical systems shown above. The best model is shown, although each trial found the same expression with a $< 1\%$ error for parameter values. The observed state variables are denoted as x and y , respectively.

The double-mass oscillator presented an additional challenge for the algorithm. When provided with the data from both masses, the algorithm found the equation of motion for each mass independently: the acceleration of one mass was proportional to the joint displacement of the masses. Theoretically, the system could be modeled from observations of a single mass using a fourth order, or jounce, differential equation. However, the data proved to be too coarse and noisy to compute accurate fourth order numerical derivatives. When provided with the data from only a single mass, the algorithm found complex third order approximations that retained a relatively high objective error and did not reflect the underlying physics.

With the addition of trigonometric operators, the algorithm was able to find the non-linear equation of motion for the pendulum. This system illustrates how expert knowledge, which in this case was the knowledge of geometry, could be leveraged to model increasingly complex systems. This concept of seeding the search with appropriate functions, and the consequences if they are absent, has been thoroughly explored in previous work [173].

These experiments on the physical systems illustrate an important potential impact for reverse engineering methods of dynamical systems. The models of all three systems converge on a common structure: the motion of these systems are dictated by second order differential equations. Through the automated mining of these systems using only the measured displacement of masses, it becomes trivial to generalize these results and extrapolate Newton’s second law of motion without any prior knowledge of physics. In comparison to other numerical approaches, these experiments exemplify how symbolic data-driven methods are capable of generating scientific insights and knowledge.

4.2.4 Conclusion and future work

A unified computational framework for the automated inference of nonlinear dynamical systems from noisy time-series that is missing least one state variable is demonstrated. By leveraging the data embedding of dynamical systems, a symbolic search was conducted for transformed representations that only contained derivatives of the observed variables in the form of implicit, ordinary differential equations of arbitrary order . These expressions were evaluated using a principle of predictive accuracy, which compared numerical derivatives with predictions obtained using a local root solver. This approach was validated on a series of synthetic and physical systems spanning a variety of qualitative behaviors, including stable equilibria, limit cycles and chaotic attractors. The method was able to consistently produce the generating equations, without any expert domain knowledge.

A current limitation of the proposed framework is that its scalability is reliant on the effectiveness of numerical differentiation of arbitrary orders from time-series data. The limitations of numerical methods, such as difficulties in calculating derivatives higher

than the fourth order, which was reflected in the experiments with the Duffing, Van der Pol and double-mass oscillators. An approach is presented that alleviates some of the scalability issues by including more state variables, which reduces the number of required derivatives and simplifies the corresponding expressions. In future work, existing active learning methods [8, 17] could complement this framework by gathering data from initial conditions that reveal more information about the missing state variables.

4.3 Discovering simple representations of dynamical systems

Most dynamical systems are only partially observable – only some variables can be directly measured, yet the overall behavior is also influenced by other hidden players that remain elusive [71, 83, 95, 193]. Although it is possible to build empirical models that are capable of accurate predictions, uncovering the hidden drivers is often the critical insight required to gain a deeper understanding into its underlying mechanisms [52, 162, 211].

For example, in predator-prey population dynamics, often only the population of a single species can be easily tracked [67, 71, 83, 95, 193]. By recording its fluctuations over time, it may be possible to predict future population sizes. However, the ability to automatically determine that the dynamics are a result of a predation interaction with an unobserved species is critical to understanding the driving factors in this system. Similar situations with hidden contributors are not uncommon in any field involving dynamical systems, ranging from chemical reactions [156] to social dynamics [29].

From a purely physical perspective, hidden state variables do not actually exist they are a matter of interpretation derived from base quantities. As a result, there are an infinite number of potential hidden state variables for a given system [50]. For example, a

swinging pendulum has an angular position that can be directly measured, but its dynamics are often described using the angular acceleration, a hidden state variable that can only be inferred indirectly by recording its position with respect to time. Furthermore, using a different set of specific hidden state variables, such as its angular momentum or kinetic energy, provides an alternative perspective in understanding the underlying system.

But how can such variables automatically be uncovered and discern them from less useful hidden state variables, such as the square root of the angular velocity? In this section, a new principle is proposed that allows for the automatic search and identification of potential hidden state variables from observations of arbitrary systems. The key idea is that useful hidden state variables reduce the overall descriptive complexity of a system. A search algorithm based on this principle is shown to successfully uncover hidden state variables in population dynamics, neuron dynamics, chemical oscillators, and chaotic systems (Fig. 4.10).

4.3.1 Related work

Finding hidden state variables can be challenging, and indeed recent methods for automatic modeling of dynamical systems from data have assumed that all variables are measured [17, 173]. There is also a variety of modeling approaches that use or estimate latent variables; however, they have a limited capacity for discovering meaningful representations. Some statistical approaches, such as hidden Markov models, are constrained to linear systems with arbitrary discrete state representations [158], while nonlinear state estimators, such as extended Kalman filters, require predefined fixed-form models de-

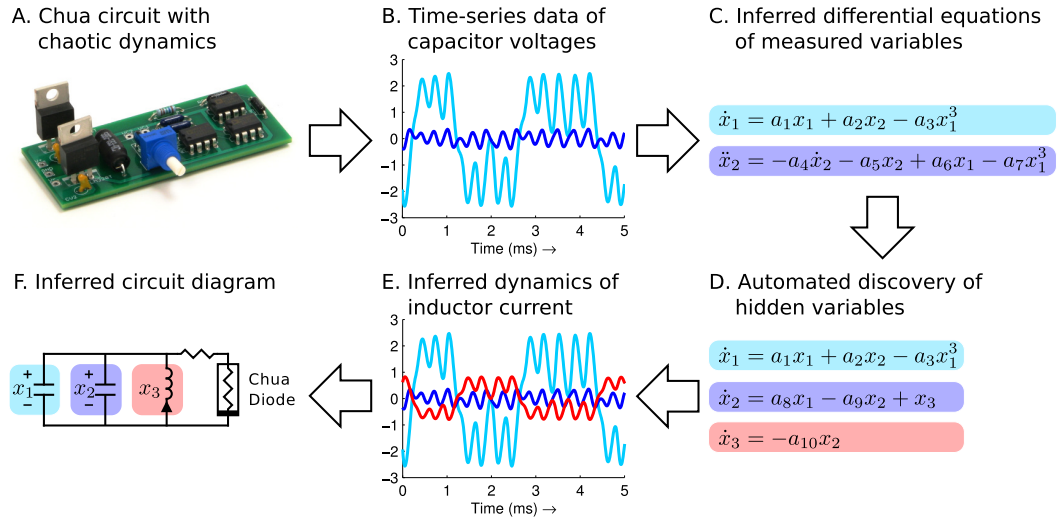


Figure 4.10: Uncovering hidden dynamical variables. (A) The voltages of the two capacitors were measured over time from a chaotic Chua circuit (B) using a digital oscilloscope. (C) The algorithm automatically searched for differential equation models that described the observed variables and (D) then uncovered transformations of the system that resulted in parsimonious realizations. (E) Without any domain knowledge of circuits or nonlinear dynamics, the algorithm found the standard formulation of the Chua circuit equations and (F) inferred the dynamics of a state variable that corresponded to unmeasured inductor current. Actual circuit, data and results are shown; inductor current is scaled linearly for visualization.

rived from expert knowledge [93]. In contrast, this algorithm seeks free-form, analytical formulations of system dynamics whose states correspond to natural quantities.

4.3.2 Inferring differential equation models of arbitrary order

This approach consists of two complementary processes (Fig. 4.11A). The first stage creates an empirical model using the observed variables in the form of an implicit, autonomous ordinary differential equation of arbitrary order. The second stage searches for new state variables, such that when the empirical model is rewritten in terms of the

candidate hidden state variable, the overall combined complexity of the description is reduced.

This approach leverages the fundamental relationships between differential equations and state space realizations of the system. For addition details, refer to Section 4.1. Although other methods, such as Takens Theorem, can determine the dimensionality of the state space through time delay analysis [197], this approach provides a powerful tool that quantifies the dynamics of the system in a representation that is suitable for finding hidden state variables.

The algorithm uses an established method to search for implicit, autonomous ordinary differential equations of arbitrary order (refer to Section 4.2). The method is based on symbolic regression [104], an established search method based on evolutionary computation [60]. Unlike traditional regression approaches that begin with equations of a prespecified form and fit unknown model parameters, symbolic regression searches for both the parameters and equation structure simultaneously. The algorithm begins by randomly combining mathematical building blocks, such as algebraic operators, constants and observed variables, to form an initial set of expressions. New candidate expressions are generated by probabilistically recombining components of previous expressions and through random variations of subexpressions. The algorithm then evaluates how well each expression models the experimental data, and retains superior models for subsequent iterations while abandoning unpromising candidates. The algorithm terminates after a desired level of accuracy is achieved and returns the set of expressions that are most likely to correspond to the underlying mechanisms of the system. For additional details, refer to Sections 1.3.2-1.3.4. Each expression is rewarded based on its ability to predict numerical relationships of higher order derivatives in the observed data.

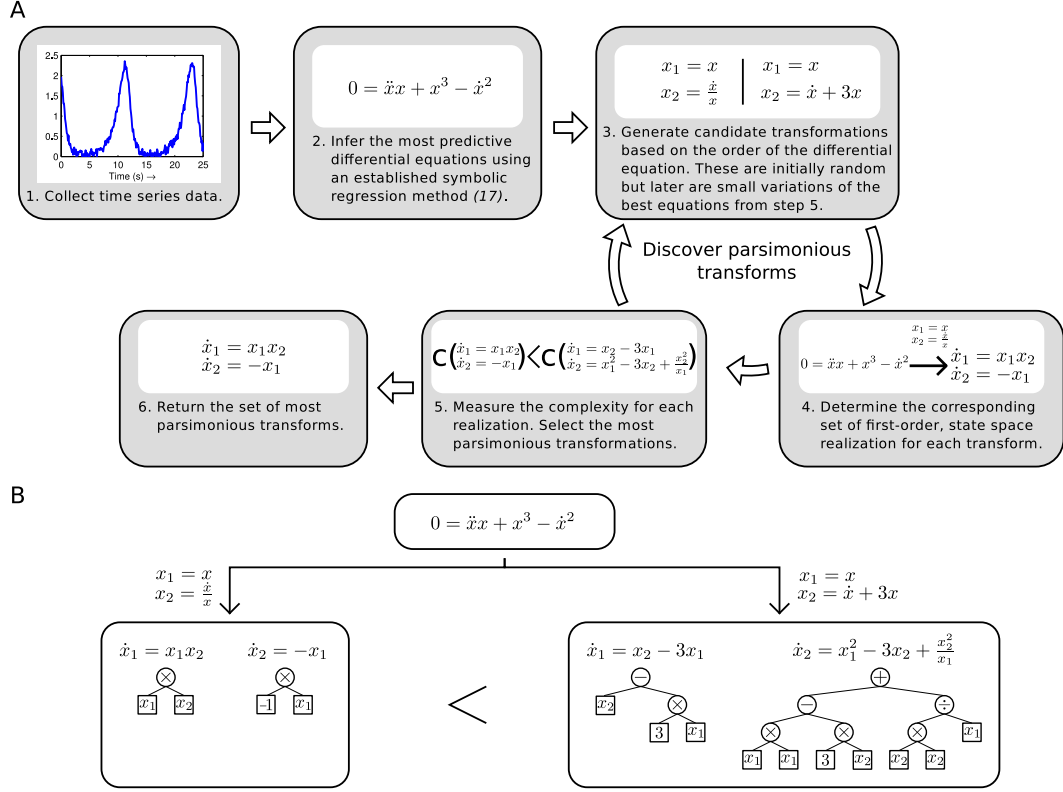


Figure 4.11: Computational approach for uncovering hidden dynamical variables from experimental data. (A) After inferring a differential equation model using the method described in Section 4.2, search for transformations that result in parsimonious realizations of the differential equation model using a symbolic regression algorithm. (B) Symbolic expressions are represented in computer memory as a tree data structure, where nodes are mathematical operations and leafs represent parameters and variables. The algorithm varies these structures to search the space of transformations. The complexity of an equation is determined by the number of elements in the corresponding tree and the sum of the coefficients weighted by their respective subtrees.

4.3.3 Inferring simple state transformations

Once a predictive differential equation model is obtained, the algorithm searches for state transformations that provide a simple and parsimonious description of the system. The complexity of an expression is quantified by a weighted measure of the number of symbols required to express the system dynamics. Promising hidden state variables are

ones that are able to describe the system using fewer terms. The approach of parsimony as a metric for suitable hidden state variables is motivated by the properties of dynamical systems. Transforming an arbitrary dynamical system into a single differential equation requires differentiation and substitution. For nonlinear expressions, differentiation is fundamentally achieved through the application of the chain rule, which inherently increases the complexity of an expression. Since the search for hidden state variables consists of essentially reversing this transformation, rewarding parsimony is an intuitive approach.

The space of possible transformations is explored using a second symbolic regression algorithm. However, unlike the first stage where expressions are selected for numerical accuracy, the algorithm searches for terse representations of the system dynamics. The system dynamics are defined by a set of coupled first-order ordinary differential equations, which can be determined from a closed form expression given a candidate transformation. The parsimony metric for each expression is defined as the size of its corresponding tree representation and the weighted magnitude of the coefficients. Though there are alternative approaches to measuring complexity, such as those based on information theory [210], this approach worked well in practice and reused much of the existing symbolic regression infrastructure. While the complexity of a single equation can be defined with a straightforward metric, the complexity of a set of coupled differential equations is more challenging to define due the fact that the complexity can be redistributed throughout the different equations.

As a result, the algorithm keeps track of the joint complexity in the set of coupled differential equations through a multi-objective Pareto optimal search, where the complexity of each equation is tracked independently and only the transformations that are not simultaneously inferior for each equation complexity are kept. Although all trans-

formations are mathematically equivalent, this multi-objective Pareto optimal approach effectively reduces the infinite space of possible transformations to a short list of approximately ten candidate hidden state variables that fit the parsimony principle. Further selection of these hidden state variable candidates requires expert domain knowledge; for example, using principles based on symmetry or non-negativity could be used to select candidate hidden state variables in population dynamics. Additionally, these candidate hidden state variables could be used as a predictive guide for the design of supplementary experiments to determine the existence of their physical counterparts; for example, new species in the environment could be tracked and compared to the candidate variables in a population dynamics study to confirm the predicted relationships.

Complexity measures of state space transformations

The second stage in finding hidden variables is using symbolic regression to search for transformations that result in parsimonious representations of the system. Note that parsimony is a poor metric for linear systems since linear equations do not get increasingly more complex due to differentiation.

First, to compare the complexities of expressions, all expressions must have the same type of representation. The expressions were compared using a sum-of-product representation, where each expression is expanded and like terms are collected. This representation was chosen since it is easy to guarantee consistency and is easy to compute through algebraic manipulation.

Unlike the fitness metric for the implicit ODE search, the fitness metric for the parsimony of a single expression is a compound vector of two terms:

$$f_{PE}(s) = [k \ p]^T \quad (4.37)$$

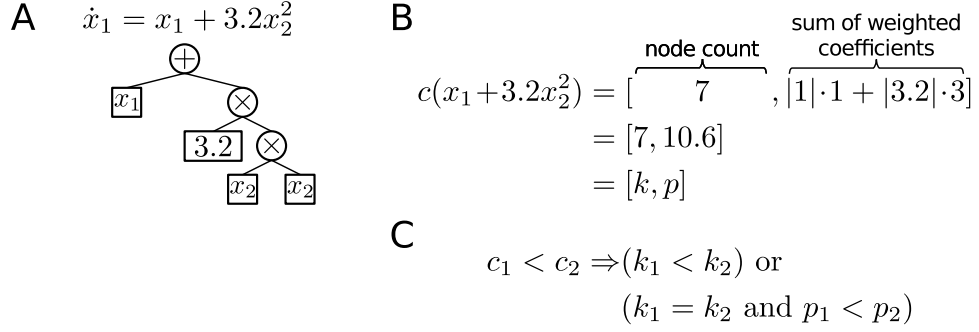


Figure 4.12: An example of calculating the parsimony fitness for an expression. For the expression and tree structure in (A), (B) illustrates how the fitness is determined: k can be interpreted as the L0-norm of the tree size while p can be interpreted as the L1-norm weighted with the corresponding L0-norm. The binary inequality is shown in (C).

where $f_{PE} \in \mathbb{NR}$ is the fitness map for parsimony of a single expression, s is the candidate transformation, k is the node complexity or the count of all the terms in the expression, and p is the sum of the absolute coefficients weighted by the node complexity of the corresponding term.

Fig. 4.12 shows an example of the parsimony for a single expression and its corresponding binary inequality. Ideally, the node complexity provides a sufficient metric for expression complexity. However, it is a poor search criterion in isolation since it provides no search gradient – the removal of nodes is a consequence of parameter values approaching zero and only counting the nodes does not encapsulate this information. Likewise, using only the coefficients is also a poor search criterion as it is possible to generate expressions with many terms with very small parameters. Instead, our fitness metric primarily values node complexity but is supplemented with the weighted sum of the coefficients to provide a search gradient.

However, the search for parsimonious representations requires a fitness metric to measure the parsimony of a transformation, not just a single expression. For a system

with n variables, the fitness metric to measure the parsimony of a transformation is:

$$f_{PR}(T(\tilde{\mathbf{x}})) = \left[k_1 \ \dots \ k_n \ \sum_{i=1}^n p_i \right]^T \quad (4.38)$$

where $f_{PR} \in \mathbb{N}^n \mathbb{R}$ is the fitness map for parsimony of a transformation, $[k_n \ p_n]^T$ is the parsimony of expression \tilde{x}_n , and $T(\tilde{\mathbf{x}})$ is the candidate transform.

Fig. S6 shows an example of the parsimony for a transformation and its corresponding binary inequality. Note that the binary inequality defines a partially ordered set as it is possible to have transformations where one expression has a greater complexity but another expression has a lower complexity. The physical interpretation of this relation is that the complexity in the system can be redistributed throughout the different equations. For example, the controllable canonical realization distributes all of the complexity into a single ODE while the remaining ODEs are trivially simple. As a result, the algorithm keeps a log of all transformations that are not dominated by any other transformation. This is implemented as a multi-objective Pareto optimal search.

Parsimonious representations arise from finding transformations that, when differentiated with respect to time, have components that are found naturally in the observable ODEs. Conceptually, this is similar to the method of integration by substitution. As a result, the simplicity in the expressions are a result of symbolic cancellation, which can occurs in Eq. 4.10 in both the substitution and matrix multiplication stages of the arithmetic processing.

Relative precision arithmetic for symbolic cancellation

A primary challenge in computing the symbolic cancellation is that the coefficients in the observable ODEs have finite precision, yet the symbolic processing needs to be sufficiently robust in canceling terms with such precision. For example, given the limited

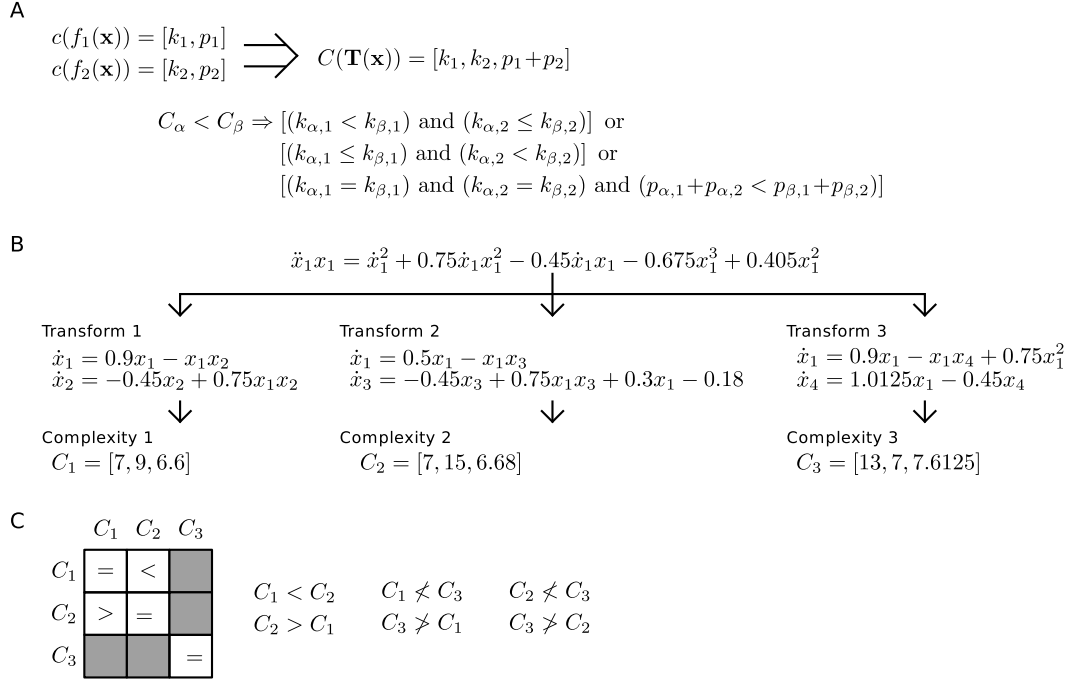


Figure 4.13: An example of calculating the parsimony fitness for a transformation. The fitness metric and its binary inequality is shown in (A). An example of the fitness for various transforms using the Lotka-Volterra dynamics is shown in (B) while a table of inequality statements is shown in (C).

precision of 4 significant digits, the expression $3 \cdot 0.3333 - 1$ should resolve to 0 and not 0.00001. Thus, the algorithm uses a method for symbolic arithmetic processing based on finite precision.

First, note that symbolic cancellation can only occur during any addition or subtraction operation. As a result, the order of operations is applied to each component in the expression with the exception of addition and subtraction. Similar terms are then collected, and the corresponding parameters are stored in a queue and sorted by magnitude. The queue is then processed in an iterative manner using relative precision from the largest to the smallest magnitudes, where the results are inserted into the queue for the next iteration, which is a common approach used in floating point arithmetic to avoid

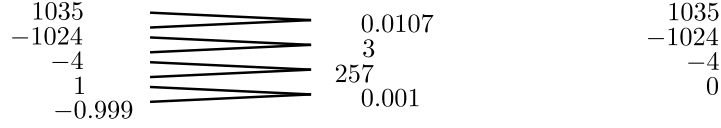
Resolve the following expression from 3 to 2 significant digits using relative precision

$$[2(345)x + x - 4x] - [2(512)x + 3(0.333)x]$$

$$= 1032x + x - 4x - 1024x - 0.999x$$

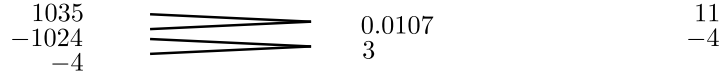
Round 0

Sort queue by magnitude → Find smallest relative precision → Resolve queue



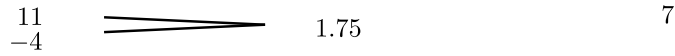
Round 1

Sort queue by magnitude → Find smallest relative precision → Resolve queue



Round 2

Sort queue by magnitude → Find smallest relative precision → Resolve queue



Solution:

$$[2(345)x + x - 4x] - [2(512)x + 3(0.333)x] = 7x$$

Figure 4.14: An example of resolving an expression using relative precision.

precision errors. The following condition is used for computing relative precision:

$$x_1 + x_2 = \begin{cases} 0 & , \text{ if } \frac{|x_1 + x_2|}{\min(|x_1|, |x_2|)} \leq 10^{-(k-1)} \\ x_1 + x_2 & , \text{ else} \end{cases} \quad (4.39)$$

where x_1 and x_2 are two parameter values, and k is the precision of the parameter values.

An example of processing an expression using relative precision is shown in Fig. 4.14.

Coefficient scaling

Finally, the search for parsimonious representations may be deceptively skewed by scaling the hidden variables with large parameters. Scaling the hidden variables may lead to false parameter gradients weighted sum of parameter values, which makes finding parameters that cancel significantly more challenging. This issue was resolved by using a heuristic approach of scale-free hidden variables where the parameter values that

correspond to the hidden variables are set to 1, if possible. This is easily achieved by taking the partial differential of the transformation and checking if the hidden variable does not appear in the resulting expression and the expression has only one term. The multiplicative inverse of the scalar is then multiplied throughout the expression to ensure that parameter in front of the hidden variable is always 1. For expressions where it was not trivial to remove the scale of the hidden variable, the expression was processed as is but it was also found empirically that these expressions were dominated in a Pareto sense with respect to expression complexity.

4.3.4 Results

The two stage approach was used to search for hidden state variables in data captured from several synthetic and physical systems (Fig. 4.15). A challenge was finding benchmark nonlinear dynamical systems with nontrivial states that could be readily confirmed. As a result, the algorithm was applied to computer-generated data for three classical nonlinear systems: Lotka-Volterra population dynamics [67, 209], Fitzhugh-Nagumo neuron model [57, 137], and the Brusselator chemical reaction [156]. Since the dynamics of these systems are known and well-established, confirming that the algorithm discovers meaningful hidden state variables is simply a matter of comparing equations. The data was generated using MATLABs Runge-Kutta methods and was corrupted with Gaussian additive noise of various magnitudes for added difficulty.

Furthermore, a physical realization of the Chua circuit [127] with a cubic nonlinearity [216] was constructed and experimental data was collected. Designed as a laboratory system that exhibits chaos as a robust physical phenomenon, the Chua circuit exhibits rich dynamics and provides a challenging modeling task. The system has three states

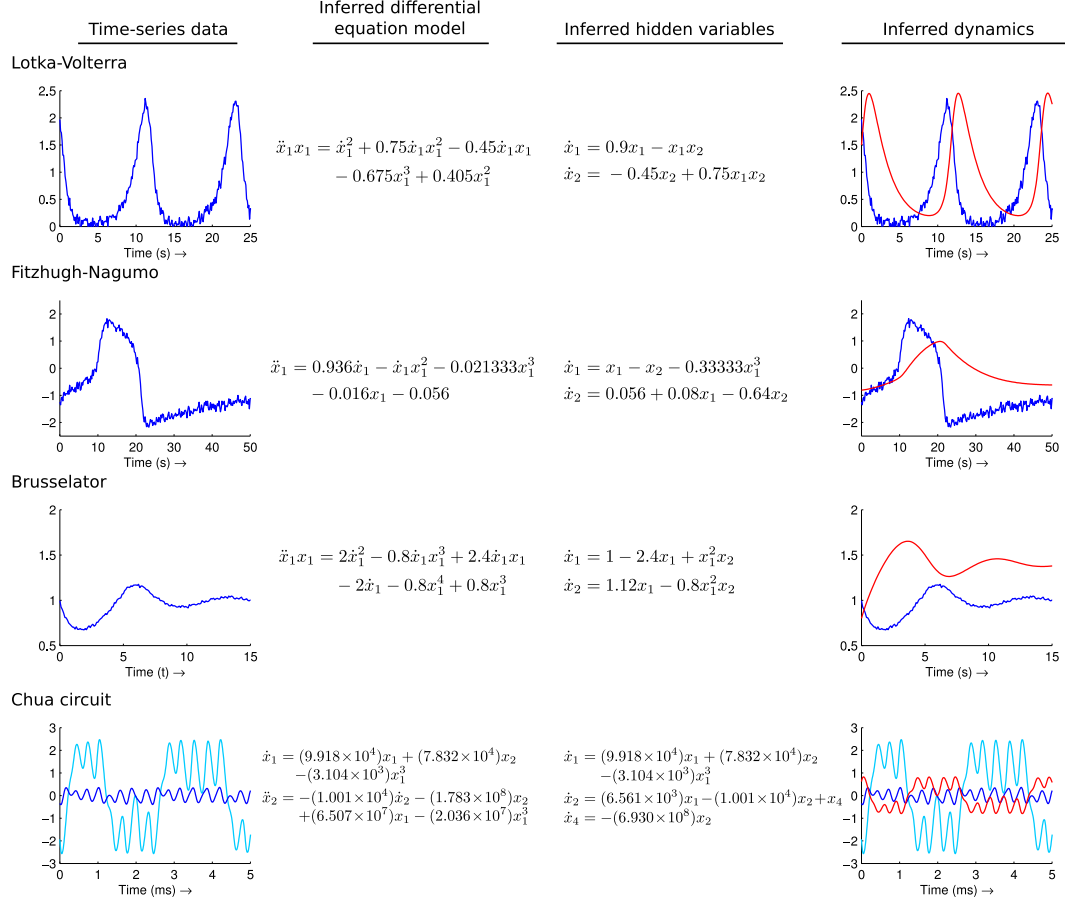


Figure 4.15: Summary of hidden variables discovered from nonlinear dynamical systems. For each system, the inferred differential equation model and inferred standard state space realization is shown, with the corresponding hidden variable as the red curve.

variables, consisting of voltages and currents, which have significance in the physical world. The cubic nonlinearity was chosen over the traditional piecewise implementation due to its smooth and more consistent behavior.

Without any domain knowledge or system models, the two stage algorithm was able to discover the existence and dynamics of hidden state variables directly from the data with missing state observations. The algorithm was able to generate the classical representations of the four systems and discovered variables that correspond to unseen population species, relaxation variables, chemical concentrations and inductor currents,

respectively. The first stage produced a collection of potential implicit differential equation models of the system, and the ground truth expression was reliably found in the accuracy-parsimony Pareto optimal set. The implicit differential models are relatively complex and finding them required a nontrivial symbolic regression search. The second stage produced a collection of approximately ten hidden state variables per implicit differential model and the classical definition of each system was found on the expression-based parsimony multi-objective Pareto optimal set. Due to the theoretical limitations of system equivalence, the hidden state variable may differ from the physical measurements by a scalar multiplicative.

The algorithm successfully discovered hidden state variables in dynamical systems with a wide range of different qualitative behaviors, including conserved trajectories, pulsing dynamics, stable orbits, and chaotic dynamics. Furthermore, the data is captured from a single trajectory of the dynamical system. Data sets that include additional trajectories with different initial conditions may yield a more complete description of the phase space and result in more robust results with faster model inference. Next, a single implementation of the algorithm was able to deal with multiple data sets where the time scales and parameter values varied in orders of magnitude with no additional modifications. The data only required minor preprocessing in the form of smoothing to minimize the effect of noise. Additional analysis, including the computational scalability and noise sensitivity of the algorithm, is available in the following subsections. Although the classical variables were found by the algorithm, the remaining hidden state variables could provide new perspectives of the system dynamics (Fig. 4.16A). In a single core implementation on a 2.8GHz Intel processor, the time required to discover hidden state variables ranged from a few minutes to tens of hours depending on the complexity of the system (Fig. 4.16B-C).

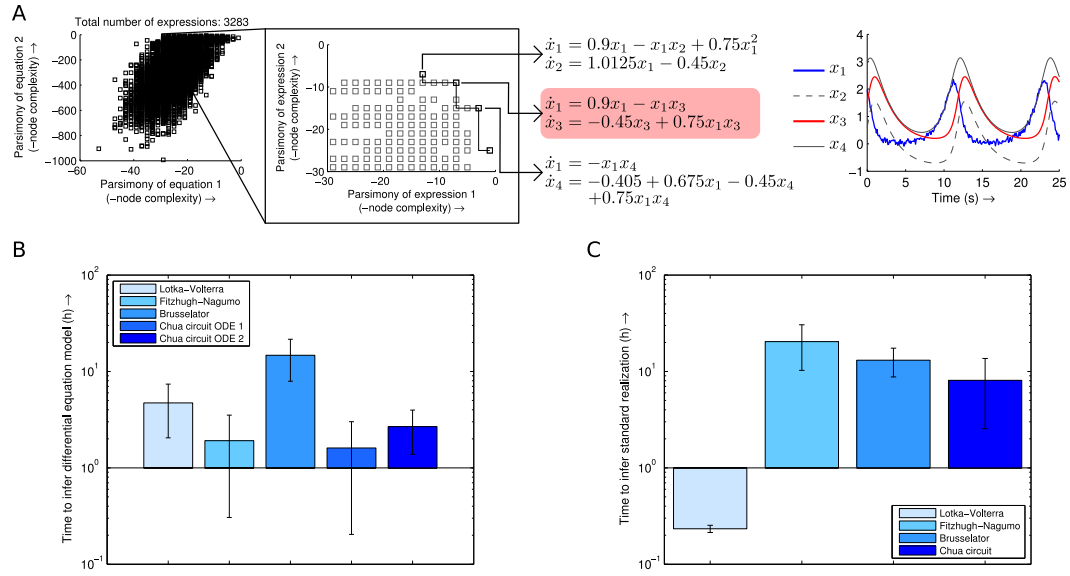


Figure 4.16: Alternative representations and computational time. (A) For the Lotka-Volterra system, the space of all transforms is searched and the set of Pareto optimal realizations with respect to parsimony are shown, with the standard state space realization highlighted in red. (B) The time to infer the differential equation model (stage 1) depends primarily on the complexity of the system and the variations in the computational time indicates how likely the algorithm finds approximate solutions. (C) The time to infer the standard realization (stage 2) depends primarily on the complexity of the transformation. Error bars indicate standard deviation ($n = 8$).

A key challenge is scaling to systems with higher complexities, and in particular, systems with larger state spaces. Inferring implicit differential equation models of higher order systems require more time-derivative variables and the resulting search space grows exponentially. The state transformation search also grows exponentially as dimensionality of the system dynamics increases. For the three state Chua circuit system, experiments with noiseless synthetic data indicate that the search algorithm is able to find the classical representations of two hidden state variables given one observed variable, but only when initialized in a small neighborhood of the globally optimal solution. For random initializations, the search settles on locally optimal solutions that numerically approximate the true dynamics, but inevitably do not lead to the discovery

of the hidden state variables. As the search algorithm does rely on heuristic methods, it is possible that the hidden state variables could be discovered with more computational effort or algorithmic advancements in the search methods.

However, an approach was proposed that incorporates additional observations to guide the discovery of hidden state variables. Even for human scientists, inferring models with hidden state variables from data is extremely challenging and this process is exponentially harder as more variables are unobserved. Instead, the algorithm is able to bootstrap and guide the experimental process using the limitations of the automated search. When the search fails to produce an accurate differential equation model, the dimension of the inferred model often remains a reliable indicator of the number of hidden state variables. Thus, higher order differential equations suggest there are several hidden state variables in the system, and despite lacking an exact description of their dynamics, the equations suggest there is a need for new experiments. Thus, the search algorithm and experiment design processes can be cycled in an iterative fashion.

This iterative approach is best illustrated with the three state Chua circuit. When only the voltage across a single capacitor is measured, the search algorithm fails to discover the other two system states. However, the inferred models are third order which guides the design of a new experiment to measure the voltage across the second capacitor. Using both observed voltage states, the algorithm is able to infer the last and unmeasured state, the current in the inductor, which provides the complete and standard description of the system. This example illustrates the potential real-world application of discovering hidden state variables since measuring currents is an experimental challenging task that cannot be done without modifying the physical circuit or changing the system dynamics. The resulting classical representation of the Chua circuit can then be used for nonlinear dynamics analysis [192].

Experimental parameters

For Stage 1: the search for implicit, higher-order observable ODEs, the algorithm described in Section 4.2.2 was used.

For Stage 2: the search for parsimonious representations, the algorithm described in Section 4.3.2 was used. The algorithm was implemented in Python, using SymPy [195] and PyParser [157] for computer algebra and equation parsing, respectively. An Age-Fitness Pareto selection method [175] was used and the operator probabilities are 5% and 70% for mutation and crossover, respectively. The encoding for each transformation is a binary tree with a maximum of 128 nodes. The operation set contains addition, subtraction, multiplication and division. The algorithm was executed on a single core on an Intel 2.8GHz processor with a population of 256 candidate transformations.

Data collection and preprocessing

The algorithm was applied to computer-generated data for three classical nonlinear systems: Lotka-Volterra population dynamics [67, 209], Fitzhugh-Nagumo neuron model [57, 137], and the Brusselator chemical reaction [156]. The data was generated using MATLABs Runge-Kutta methods and Table 4.3 details the integration parameters.

Furthermore, each system was corrupted with Gaussian additive noise of various magnitudes for added difficulty. Three conditions were generated for each system: noiseless, 1% and 10% of the standard deviation of the signal. A local polynomial regression fit (S22) was then used for noise rejection with 10-fold cross validation.

Finally, a physical realization of the Chua circuit [127] with a cubic nonlinearity was constructed based on the model outlined in [216]. The system has three states variables,

Table 4.3: The parameters for generating the synthetic systems

System	Equations	Initial Conditions	MATLAB Function	Integration Tolerances
Lotka-	$\dot{x}_1 = 0.9x_1 - 0.5x_1x_2$	$x_1(0) = 2$	ode23	(1e-3, 1e-3)
Volterra	$\dot{x}_2 = -0.45x_2 + 0.75x_1x_2$	$x_2(0) = 2$		
Fitzhugh-	$\dot{x}_1 = x_1 - \frac{1}{3}x_1^3 - x_2$	$x_1(0) = -1.994$	ode45	(1e-6, 1e-6)
Nagumo	$\dot{x}_2 = 0.08(x_1 + 0.7 - 0.8x_2)$	$x_2(0) = 0.81$		
Brusselator	$\dot{x}_1 = 1 - 2.4x_1 + 0.8x_1^2x_2$	$x_1(0) = 1$	ode45	(1e-8, 1e-8)
	$\dot{x}_2 = 1.4x_1 - 0.8x_1^2x_2$	$x_2(0) = 1$		

consisting of voltages and currents, which have significance in the physical world. The cubic nonlinearity was chosen over the traditional piecewise implementation due to its smooth and more consistent behavior. The identical Analog Devices AD633JN multipliers and AD711KN operational amplifiers were obtained. The circuit was powered by LM340T 15V and LMT320 -15V voltage regulators. The resistors, capacitors and inductors were matched as closely to the original specifications based on available components, with a maximum of 15% difference in value. The primary resistor in the circuit was replaced with a linear potentiometer, which was tuned by hand until chaotic dynamics were observed. The voltages across both capacitors were measured using an Agilent DSO-X 3034 digital oscilloscope.

Inferring implicit ODEs of synthetic systems

Figure 4.17 summarizes the Stage 1: inferring implicit, higher-order observable ODEs for the three synthetic systems. The performance of the algorithm is described by its fitness as a function of computational effort and the corresponding fitness-parsimony

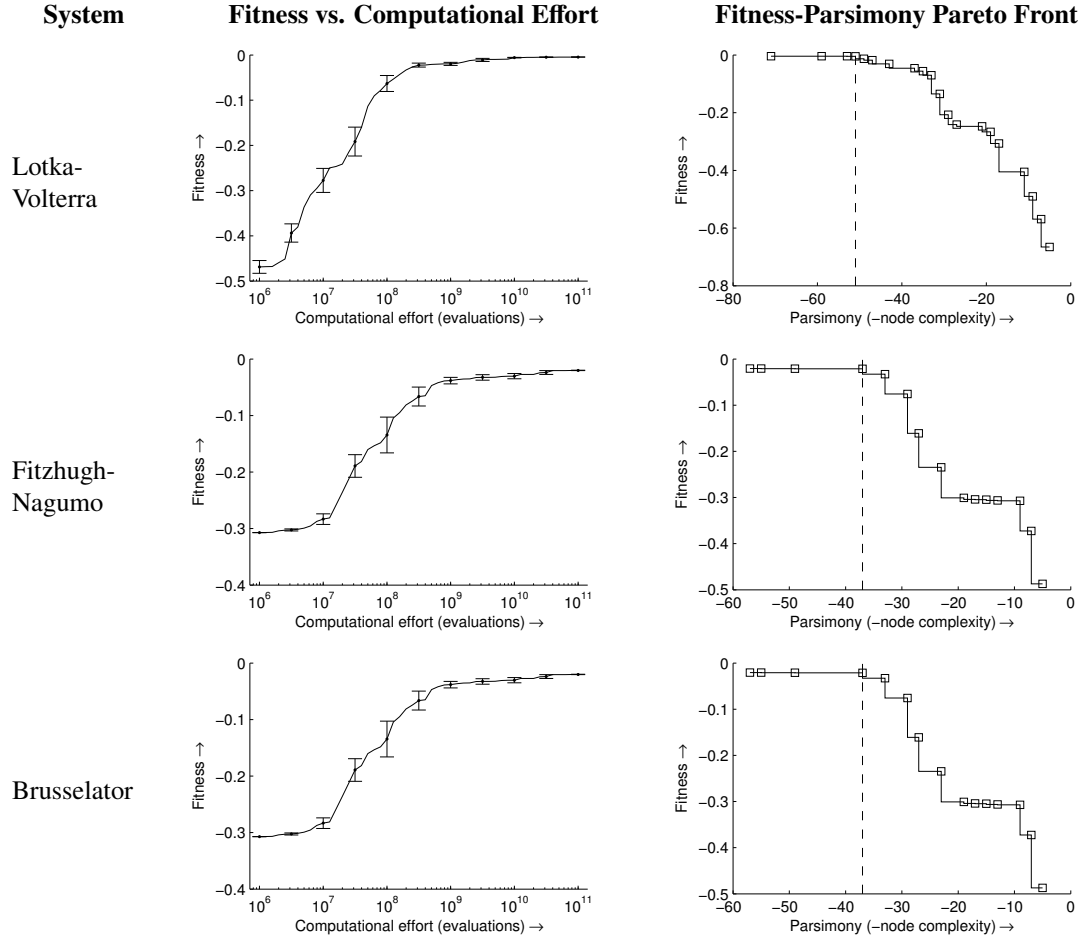


Figure 4.17: The performance plots for the Stage 1 inference of synthetic systems. Error bars indicate standard error ($n = 8$).

Pareto front. The dashed line in the Pareto fronts indicate the complexity of the differential equation model - any model with greater complexity achieves only marginal improvements in fitness, while any model with lower complexity obtains significantly poorer fitness. Computational effort is measured by the number of expression evaluations, which is a robust metric as it is independent of implementation and runtime.

Figure 4.18 summarizes the impact of noise on the inference problem. In particular, noise makes numerical differentiation significantly more difficult. A local polynomial fit was used with 10-fold cross validation to remove high frequency noise from the system.

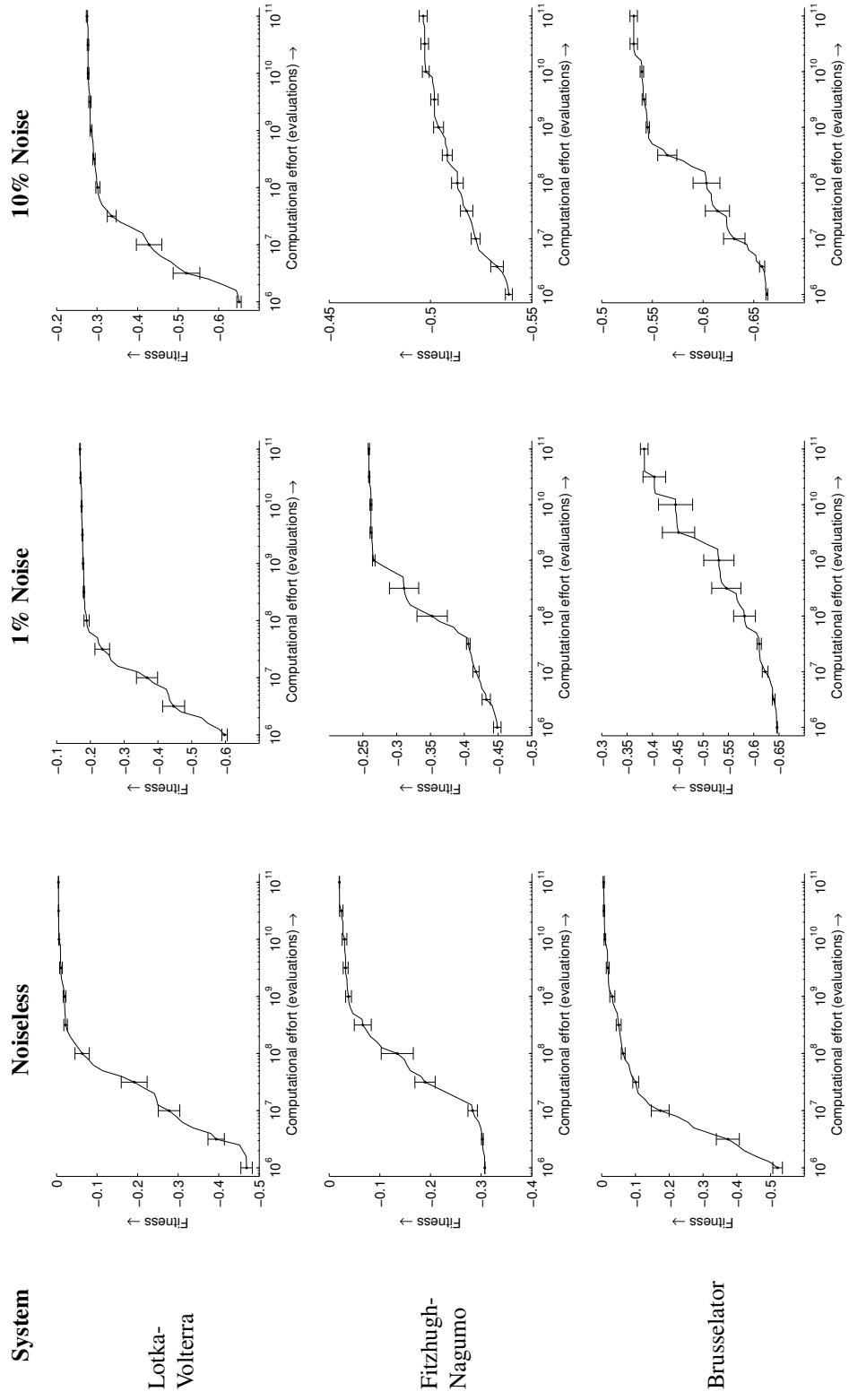


Figure 4.18: The fitness plots as a function of noise. Error bars indicate standard error ($n = 8$).

Inferring parsimonious representations of synthetic systems

The performance of the Stage 2: inferring parsimonious representations algorithm is significantly more challenging to visualize since there is no monotonic fitness metric. Rather than focusing attempting to optimize a single goal, the algorithm has an exploratory nature that attempts to find as many parsimonious representations as possible. Thus, it is more meaningful to show the space that the algorithm has explored within a computational limit. Figure 4.19 shows a plot of the search for each system, where each point represents at least one expression for a given complexity.

Figure 4.20 shows the most parsimonious, nontrivial representations found by the search algorithm along with their complexity.

Figure 4.21 shows a plot of the most parsimonious, nontrivial representations found by the search algorithm with respect to the original data set.

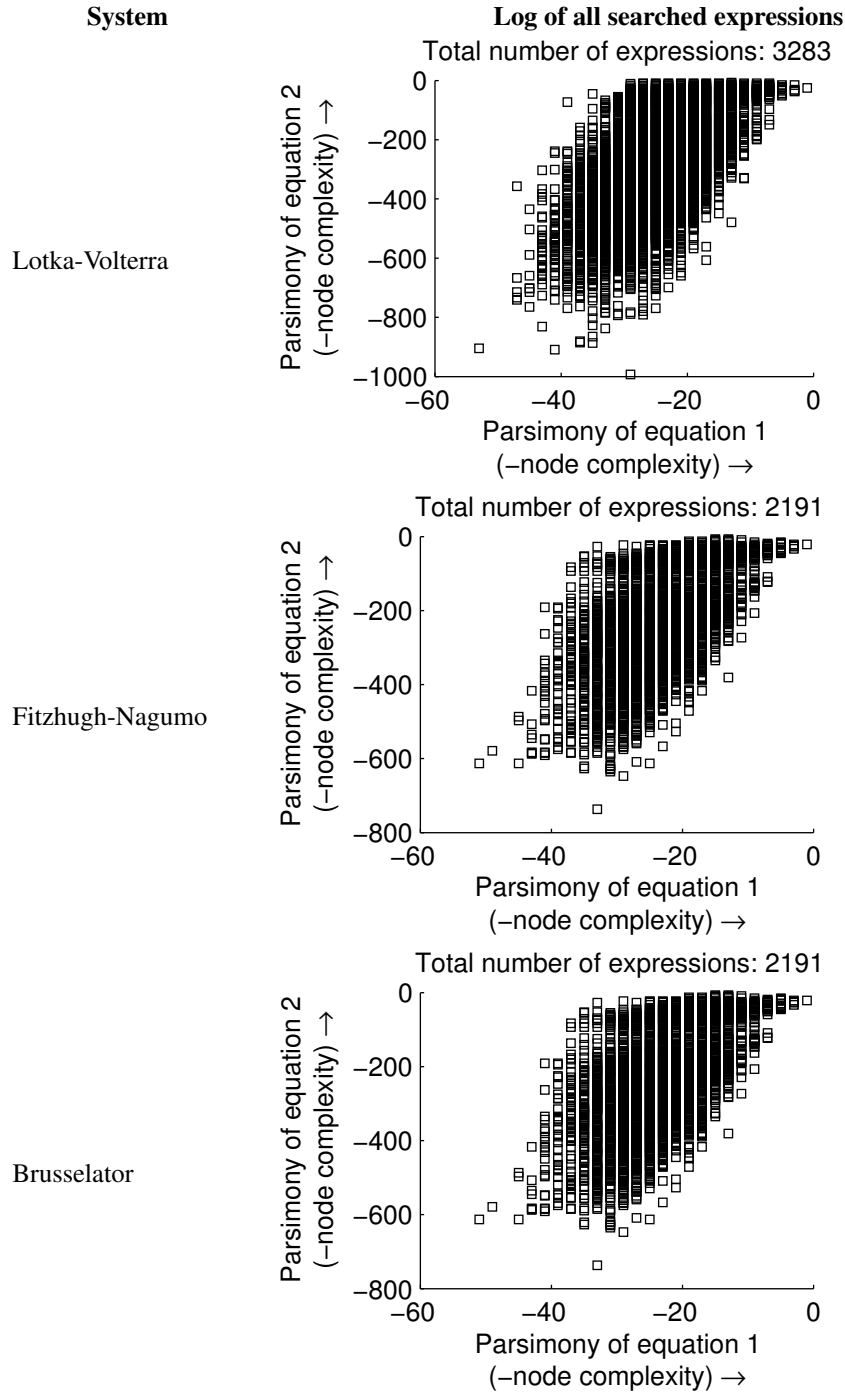


Figure 4.19: The search log plot for each synthetic system.

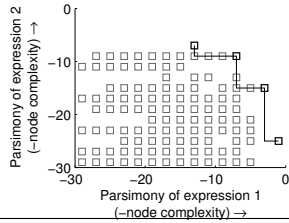
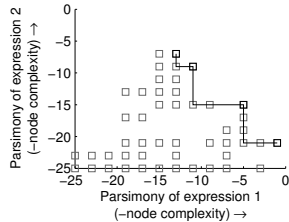
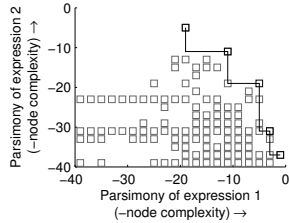
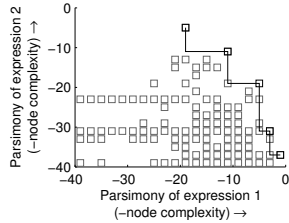
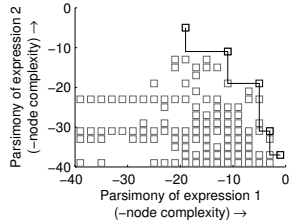
System	Parsimony Pareto Front	Realizations	Complexity
Lotka-Volterra		$\dot{x}_1 = -x_1 x_2$	[3, 15, 6.78]
		$\dot{x}_2 = -0.405 + 0.675x_1 - 0.45x_2 + 0.75x_1 x_2$	
		$\dot{x}_1 = 0.9x_1 - x_1 x_3$	[7, 9, 6.6]
		$\dot{x}_3 = -0.45 + 0.75x_1 x_3$	
Fitzhugh-Nagumo		$\dot{x}_1 = -0.21333x_1 - x_2$	[5, 15, 7.0710]
		$\dot{x}_2 = 0.056 + 0.95733x_2 + 0.036423x_1 - x_1^2 x_2$	
		$\dot{x}_1 = x_1 - x_2 - 0.33333x_1^3$	[11, 9, 3.1093]
		$\dot{x}_3 = 0.056 + 0.08x_1 - 0.64x_3$	
Brusselator		$\dot{x}_1 = -0.875 + x_1 - x_4 + 0.033333x_1^3$	[13, 7, 3.9283]
		$\dot{x}_4 = 0.08x_1 - 0.64x_4$	
		$\dot{x}_1 = x_1 x_2$	[3, 31, 21.6]
		$\dot{x}_2 = 0.8x_1 + 2.4x_2 - 0.8x_1^2 x_2 - 0.8x_1^2 + x_2^2 - 2\frac{x_2}{x_1}$	
Brusselator		$\dot{x}_1 = x_1^2 x_3$	[5, 19, 18.2]
		$\dot{x}_3 = 0.8 + 2.4x_3 - 0.8x_1^2 x_3 - 2\frac{x_3}{x_1}$	
		$\dot{x}_1 = 1 - 2.4x_1 + x_1^2 x_4$	[11, 11, 13.52]
		$\dot{x}_4 = 1.12x_1 - 0.8x_1^2 x_4$	
Brusselator		$\dot{x}_1 = 1 - 2.4x_1 + x_1^2 x_5 - 0.8x_1^3$	
		$\dot{x}_4 = 0.8 - 0.8x_1$	[19, 5, 14]

Figure 4.20: The most parsimonious, nontrivial representations found by the search algorithm.

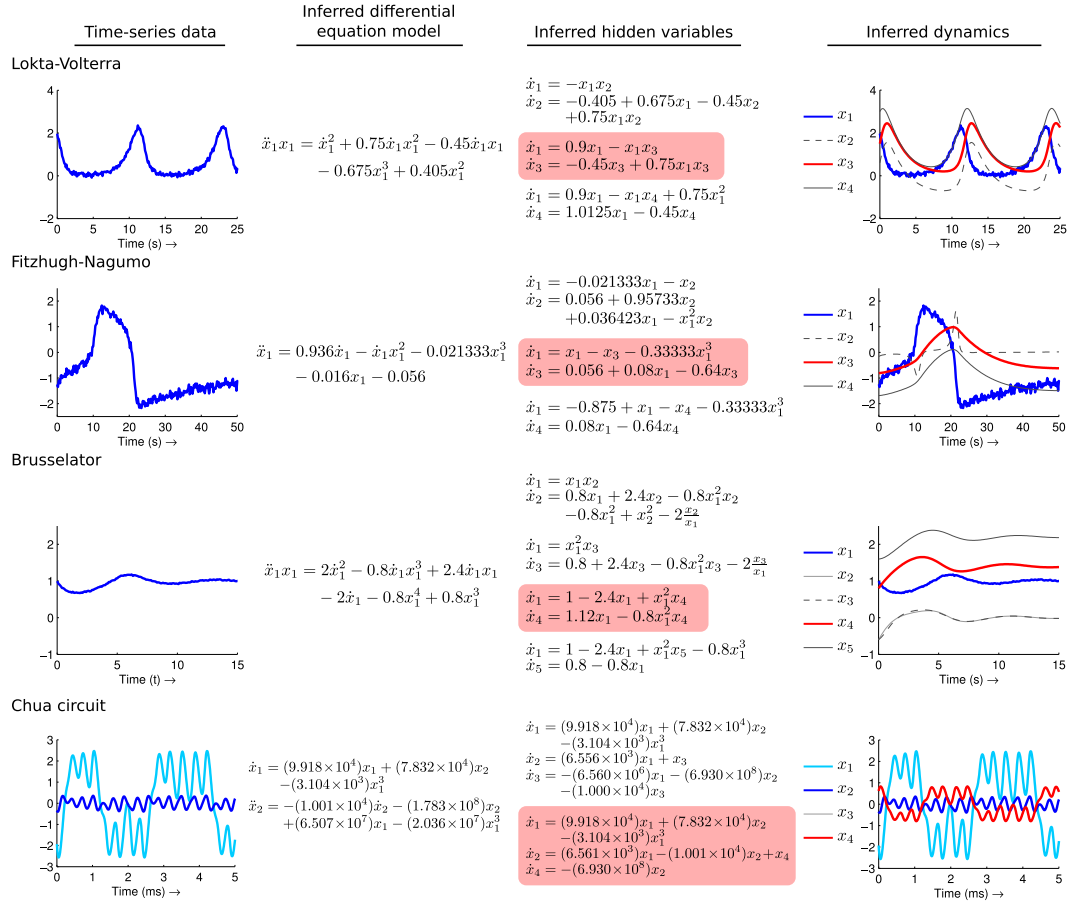


Figure 4.21: A plot of the most parsimonious, nontrivial representations found by the search algorithm for each dynamical system. The standard realizations are highlighted in red.

Chua circuit inference

This section outlines the performance results of discovering hidden variables for the Chua circuit as outlined in Section 4.1.4. Figure 4.22 shows the fitness over computational effort and the fitness-parsimony Pareto fronts for the Stage 1 performance. The Stage 2 performance is outlined in Figure 4.23, which shows the log of all transformations searched, and in Figure 4.24, which summarizes all the most parsimonious, nontrivial transformations.

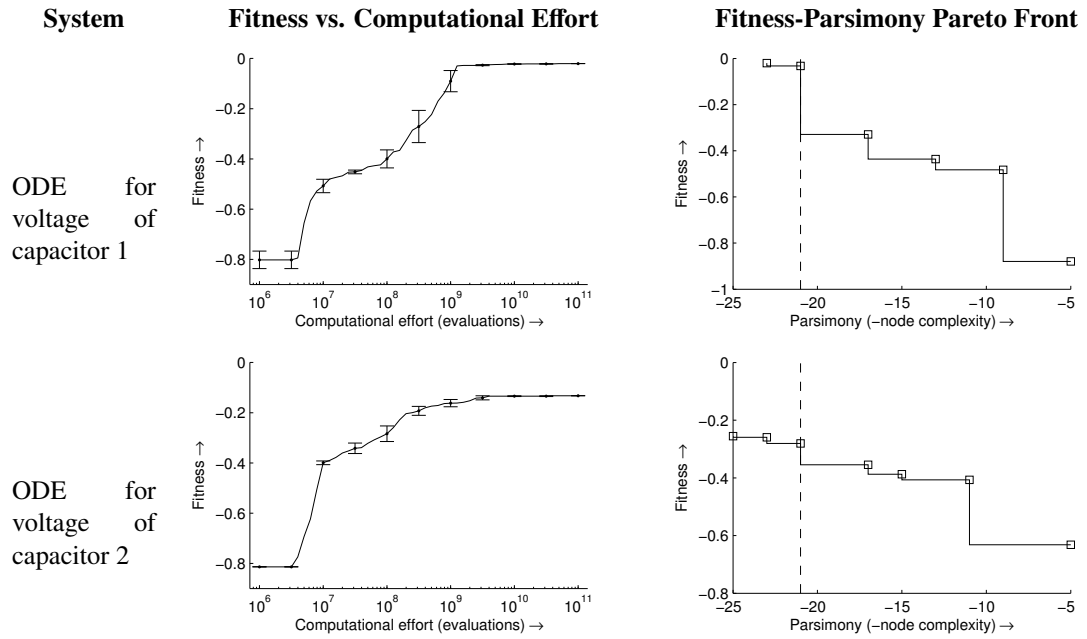


Figure 4.22: The performance plots for the Stage 1 inference of the Chua circuit. Error bars indicate standard error ($n = 8$).

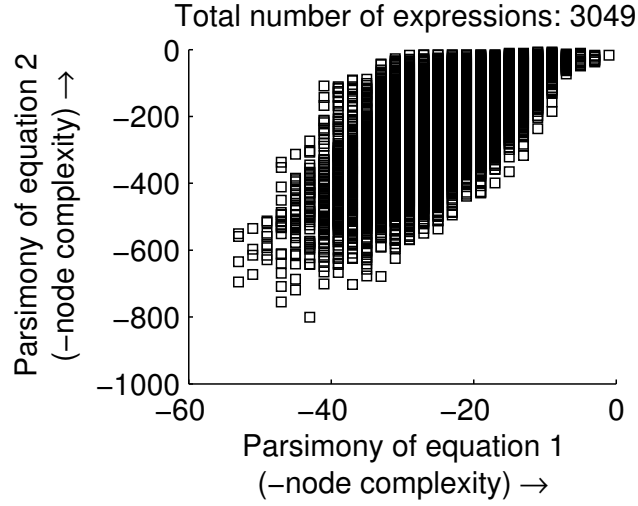


Figure 4.23: The search log plot for the transformations in the Chua circuit.

Parsimony Pareto Front	Realizations	Complexity
	$\dot{x}_1 = (9.918 \times 10^4)x_1 + (7.832 \times 10^4)x_2 - (3.104 \times 10^3)x_1^3$ $\dot{x}_2 = (6.556 \times 10^3)x_1 - x_3$ $\dot{x}_3 = -(6.560 \times 10^6)x_1 - (6.930 \times 10^8)x_2 - (1.000 \times 10^4)x_3$	[15, 5, 11, 6.997×10^8]
	$\dot{x}_1 = (9.918 \times 10^4)x_1 + (7.832 \times 10^4)x_2 - (3.104 \times 10^3)x_1^3$ $\dot{x}_2 = (6.561 \times 10^3)x_1 - (1.001 \times 10^4)x_2 + x_4$ $\dot{x}_4 = -(6.930 \times 10^8)x_2$	[15, 9, 3, 6.932×10^8]

Figure 4.24: The most parsimonious, nontrivial representations of the Chua circuit found by the search algorithm.

Computational effort

Figure 4.25 summarizes the computational effort and equivalent clock runtime required to infer the differential equation model for each synthetic system while Figure 4.26 summarizes the computational effort and equivalent clock runtime required to infer the standard realization each system. Note there is a nonlinear mapping between expression evaluations and computational effort, as the time it takes to evaluate a function depends on its complexity.

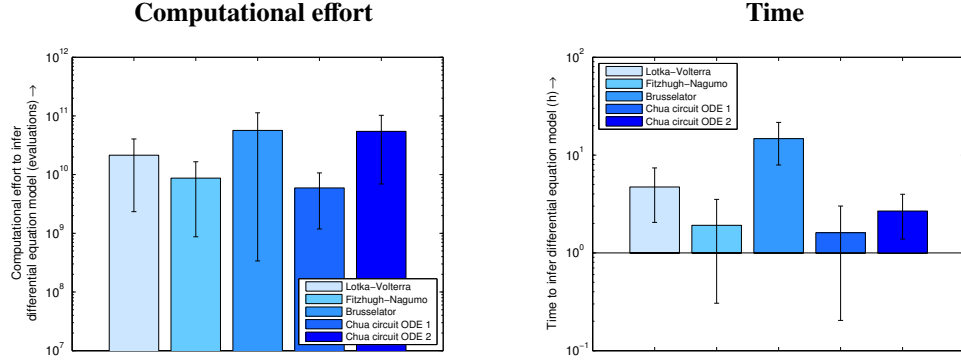


Figure 4.25: The computational effort and clock time required to infer the differential equation model. Error bars indicate standard deviation ($n = 8$).

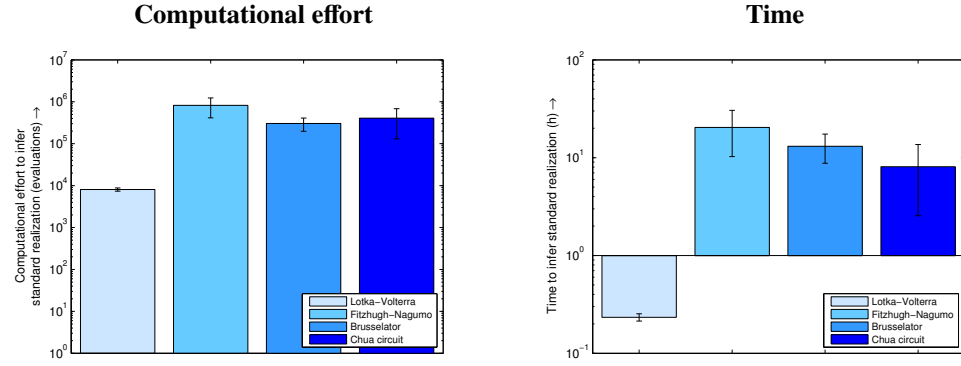


Figure 4.26: The computational effort and clock time required to infer the standard realization. Error bars indicate standard deviation ($n = 8$).

4.3.5 Conclusions and future work

The discovery of hidden state variables was demonstrated from a variety of nonlinear dynamic systems directly from experimental data with the use of a computational search. The algorithm is based on a two stage approach that was able to infer the existence and dynamics of hidden variables in population dynamics, neuron dynamics, chemical oscillators, and chaotic systems without any prior knowledge about any of these diverse fields. The exact classical representations of these systems, which include the dynamics

of unseen variables, were reproduced by the algorithm. The automated discovery of hidden state variables has the potential of accelerating the scientific process and guiding scientists in understanding the underlying mechanisms in complex, natural phenomena.

4.4 Modeling spatial differential systems

This section presents preliminary research on modeling spatial and temporal differential systems using partial differential equations. Partial differential equations are expressions that relate multivariable functions and their partial derivatives. Many important physical phenomena are formulated using differential equations, including classical mechanics, the propagation of heat and waves, electrodynamics and fluid flow. While many physical phenomena are qualitatively different, upon further inspection, their differential equations indicate that they are actually governed by the same underlying principles. Consequently, the ability to infer differential equations directly from data, rather than relying on first principle derivation, provides a critical tool for automated scientific discovery.

This section builds directly on the approaches presented in Section 4.2.2, since theoretically, it is a general framework for inferring differential equations of arbitrary order using symbolic regression and partial differential equations are a subset of this general task.

4.4.1 Symbolic representation

To find implicit ordinary differential equations, symbolic regression [104], an established search method based on evolutionary computation [60], is used. Unlike traditional

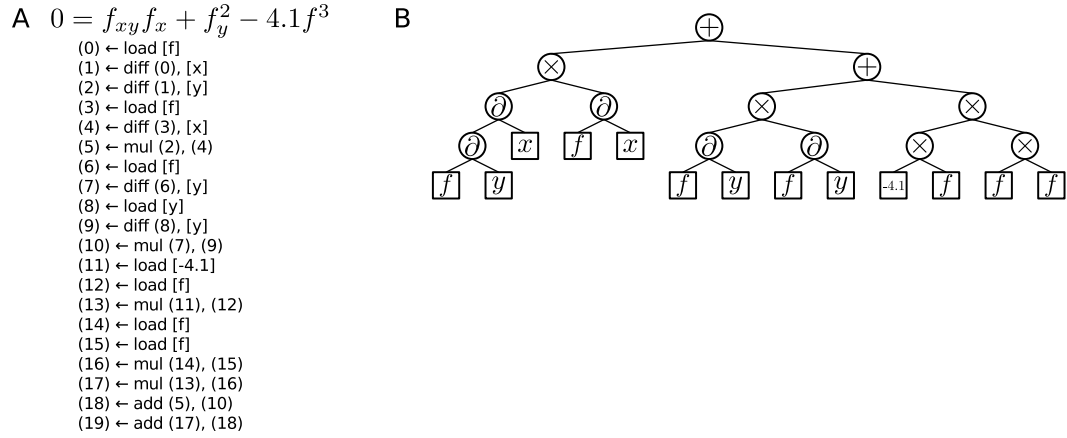


Figure 4.27: The representation of a partial differential equation model in computer memory (A) and its corresponding tree representation (B).

regression approaches that begin with equations of a prespecified form and fit unknown model parameters, symbolic regression searches for both the parameters and equation structure simultaneously. The algorithm begins by randomly combining mathematical building blocks, such as algebraic operators, constants and observed variables, to form an initial set of expressions. New candidate expressions are generated by probabilistically recombining components of previous expressions and through random variations of subexpressions. The algorithm then evaluates how well each expression models the experimental data, and retains superior models for subsequent iterations while abandoning unpromising candidates. The algorithm terminates after a desired level of accuracy is achieved and returns the set of expressions that are most likely to correspond to the underlying mechanisms of the system. For additional details, refer to Section 1.3.2.

For partial differential equations, expressions are represented as binary trees that consist of algebraic operations with numerical constants and symbolic variables at its leafs. Using a similar representation to the ordinary differential equation, partial differential equations are modeled using building blocks of addition, multiplication, differentiation, constants as well as independent and dependent variables. There is a syntactic

restriction that only allows differentiation blocks to differentiate dependent variables and other differentiation blocks with respect to the independent variables. Examples of this representation is shown in Fig. 4.27.

4.4.2 Fitness evaluation

Defining a suitable method to evaluate fitnesses for partial differential equations proved to be the fundamental challenge in this research due to a variety of reasons. First, non-linear partial differential equations lack many of the same properties that are found in ordinary differential equations. In particular, there is no guarantee of existence and uniqueness of solutions [74]. For ordinary differential equations, many of the methods relied on repeatedly solving the initial value problem for an arbitrary equation and comparing the solution with the data. The natural extension of this technique to partial differential equations is to solve boundary value problems for an arbitrary equation and comparing the solution with the data. However, from a theoretical perspective, the lack of existence and uniqueness means there is no guarantee that a convergent solution can be obtained for an arbitrary expression.

However, the more limiting constraint in practice was that achieving numerical integration for an arbitrary partial differential equation was difficult. The most popular technique for solving partial differential equations are the finite element methods, which relies on variational methods to minimize an error function and produce a stable solution. This requires iteratively modifying each element within a larger mesh to reduce the error of the solution. This approach is extremely computationally expensive and, since evolutionary computation requires many evaluations, is unsuitable as a underlying method for fitness evaluations.

A simpler approach consists of using finite difference methods, such as the Crank-Nicolson scheme [39]. A limitation in these approaches is automating the stencil for arbitrary partial differential equations. Furthermore, while these methods are numerically stable, the growth in error often leads to unusable results. These approaches are discussed in detail in the following subsection.

Integration via finite difference stencil. The initial approach for an objective error metric was based on numerical integration with a finite difference method. Each differential operator was expanded using its finite difference representation and each data point was solved using a local root finding algorithm. Since the approach used numerical integration, the data points were used as a boundary condition and the algorithm integrated until a user defined error was reached. The fitness was the summed count of the integration steps as well as the residual error. For implementation details, refer to Algorithm 4.4.

This approach suffered from a number of numerical accuracy. Even for small timesteps, the growth in error resulted in unusable solutions for comparison with the measured data. Using different stencils, such as implicit methods, explicit methods and Crank-Nicolson methods [39], only had a minor effect on the results. Even simple linear partial differential equations, such as the heat equation or wave equation, proved to be a challenge when the temporal or spatial timesteps increased. Furthermore, finite difference methods had no ability to deal with noisy boundary conditions.

Integration via black-box, adaptive Runge-Kutta methods. Following the process from Section 4.2.2, it would have been ideal to formulate an automated method using a higher-order adaptive stepsize method, such as the Runge-Kutta methods. However,

Algorithm 4.4: Integration via finite difference stencil

```
1  input  → expression
2  output → fitness value = [count overflow]T
3
4  function eval_expression(expr, input_value, data, data_index) :
5      fd_expr = replace_derivative_with_finite_difference(expr)
6      numerical_expr = substitute_fd_using_data(data, data_index)
7      return resolve_expression(numerical_expr, input_value)
8
9
10 function secant_method(expr, data, data_index) :
11     pred_2 = data[data_index-2]
12     pred_1 = data[data_index-1]
13     pred   = 0
14
15     for i in range(secant_method_limit) :
16         expr_value_2 = eval_expression(expr, pred_2, data, data_index)
17         expr_value_1 = eval_expression(expr, pred_1, data, data_index)
18         if (expr_value_1 = expr_value_2) :
19             break
20         else :
21             pred = pred_1 - expr_value_1*
22                 (pred_1 - pred_2)/(expr_value_1 - expr_value_2)
23             pred_2 = pred_1
24             pred_1 = pred
25     return pred
26
27
28 fitness.count = 0
29 fitness.overflow = 0
30
31 stdev = get_highest_order(expr)
32 for each dimension :
33     for i in range(data) :
34         error = 0
35         count = 0
36         while (error < user_limit) :
37             prediction = secant_method(expr, data, i)
38             error += abs(data[i] - prediction)
39             count += 1
40
41         fitness.count += count
42         fitness.overflow += fitness.error/stdev/range(data)
43
44 return fitness
```

translating the one-dimensional approach of ordinary differential equations to a multi-dimensional partial differential equation proved to be challenging and was abandoned.

Comparing predicted derivatives. The final approach, which an objective error metric that compares the values of the highest order differential using two numerical ap-

proaches, was not thoroughly implemented for partial differential equations but may prove to have the most potential. Effectively, this approach converts the implicit equation into an explicit equation for the highest order differential using numerical methods. The first method for calculating the highest order differential determines the value by numerical differentiation methods, while the second method for calculating the highest order differential substitutes the numerical derivations of the lower order differentials into the candidate expression and solves for the highest order differential using a local root finding method.

Unlike the previous approaches that relied on determining a solution by numerical integration, which is difficult for partial differential equations, this approach relied only on local information. Thus, it is significantly easier to calculate partial derivatives and compare them numerically on a local basis. Also, the implementation is simple and should be much faster to compute than the other methods. This is a prime area for future research.

CHAPTER 5

AUTOMATED TELESCIENCE

The scientific process is often laborious and time-consuming. Although it is easy to identify the existence of a phenomenon, providing a rigorous explanation requires systematic experimentation and detailed characterization. The primary goal for the field of automated science is to uncover the governing principles of natural phenomena automatically merely by perturbing and observing a system in an intelligent fashion, just as a scientist would do with an experimental system. However, determining the full internal structure of unknown systems is particularly challenging when subtle but critical behavior can go unobserved unless the system is perturbed in very specific ways. Coarse parameter sweeps or random samplings are unlikely to catch these subtleties, and so passive machine learning methods are likely to be ineffective.

The field of automated science aims to both automate as well as accelerate this discovery process by automating the cyclic components of hypothesis-formation and experimentation. Hypothesis-formation consists of two active learning approaches: the first algorithm explores and formulates candidate models that explain the observed data, while the second algorithm selects the experiment that best disambiguates the competing models. The experimentation component is executed by a phenomenon-specific instrument that is able to elicit and record a range of different behaviors by following a programmable procedure. The accelerated discovery process arises from the optimal experiment design as well as the ability to work ceaselessly without fatigue. The process of automated telescience is summarized in Fig 5.1.

A key characteristic of automated science is that it completely disembodies the scientific process, allowing for scalability via remote operation or automated telescience. Automating each of the two components, physical experimentation and hypothesis-

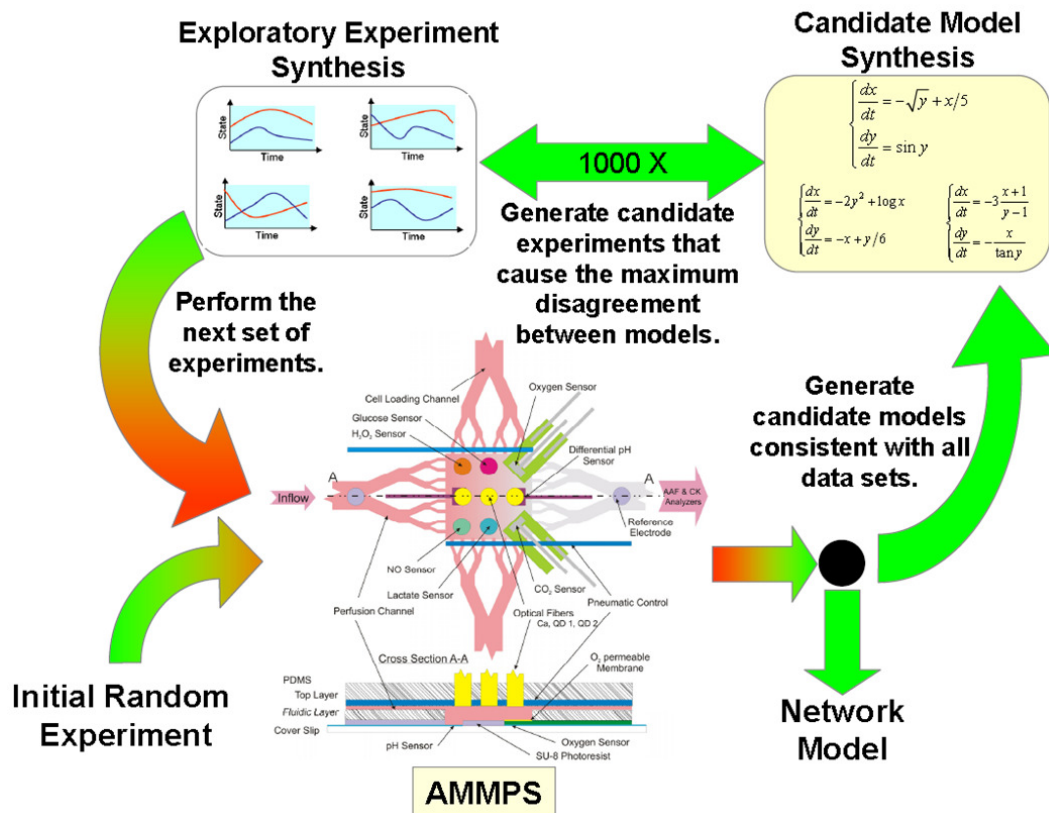


Figure 5.1: A flowchart illustrating the major components of automated tele-science. The cyclic processes of hypothesis-formation and experimentation are shown. Image courtesy of John Wikswo.

formation, requires domain-specific knowledge and expertise. However, this modular nature is well-suited for coordinated development with specialized institutions – the phenomenon-specific instrument is developed by one group of experts and the active learning algorithms are developed by another. Through the remote operation via telecommunication, this decoupled approach overcomes the geographic constraints of such multidisciplinary projects.

An unexpected consequence of remote experiments is that many traditional properties no longer guaranteed. In local experiments, the environment and instrument status are supervised, and thus, any obvious external disruptions, such as inadvertent colli-

sions, are identified immediately and dealt with accordingly. However, with remote experiments, a robust protocol must be developed ensure that properties such as repeatability and stationarity are maintained throughout the data collection process.

The aim of this work is to design automated systems that are capable of inferring scientific principles. The opportunities for automated telescience are near limitless – it can be used to accelerate the investigation of any phenomenon with the design of the appropriate experimental instrument. This data-driven model inference is ideal for disciplines dominated by data-rich experimentation such as pharmaceutical engineering, genetic engineering, materials engineering and energy systems.

This chapter begins with a discussion of related work in Section 5.1. The chapter is then divided into two sections, which outlines work on two separate projects: Section 5.2 discusses work on a project to automatically infer bioluminescent chemical reactions while Section 5.3 discusses work on a project to discover dynamical models of a microfluidic chamber, as an physical analogy for multi-compartment pharmacokinetic models, using a remote controlled robotic device. Both of these projects were done in collaboration with the Vanderbilt Institute of Integrative Biosystems Research and Education at Vanderbilt University led by Professor John P. Wikswo.

5.1 Related work

The concept of automating the scientific process is a relatively new idea made possible by the advancements of computer science and the increasing availability of computational power [105, 107]. As a result, the number of projects in this field have been relatively limited. Early attempts at automated science include: DENDRAL [115], Meta-DENDRAL [26, 55], Automated Mathematician [110], EURISKO [110], KEKADA,

ABACUS [53], BACON [108], Fahrenheit [217] and IDS [139]. These early approaches often processed a static data set and did not design or execute their own experiments through active interactions with their environments. These projects are summarized more thoroughly in Section 1.2 and a survey of this topic is reviewed by Sparkes et al. [189].

Recent work in this field has achieved significant developments in the areas of active learning and automated interactions for scientific discovery. The most prominent example of this work was a robotic system named ‘Adam’, which was able to identify genes encoding ‘locally orphan enzymes’ in the yeast *Saccharomyces cerevisiae* [98]. The automated laboratory robotic system used formal logical expressions to analyze a bioinformatic database and homology search techniques to hypothesize likely candidate genes for discovery. The experiment design code used a system model to generate biological experiment plans, which were executed on microplate layouts and passed to the robotic system for processing. The resulting growth curve data is processed using a spline fitting post-processing, and predetermined biologically significant parameters such as growth rate and lag time were extracted. The parameters from multiple experiment replicates were analyzed to either confirm or refute the hypothesis.

Historically, there has also been a significant amount of research on optimal experiment design for biological systems. Much of the early work focused on optimal experiment design between two competing prespecified models as statistical models [7, 19, 84]. More recent work extended this approach to nonlinear [33, 59] and dynamical systems [201]. A survey of this topic is reviewed by Fedorov [54].

The work presented in this chapter extends the automated science framework in two key areas. First, this work aims to learn more general scientific models and discover unknown structure. In comparison with the ‘Adam’ robot, which used well-structured

analysis to extract parameters that were known in advance to be significant, this research aims to build general dynamical models that capture the underlying structure of the system. Second, both the physical robotic device as well as the learning algorithms are located in the same room for the ‘Adam’ robot. This work investigates the challenges of remote controlled scientific discovery where many of the traditional experimental properties can no longer be guaranteed.

5.2 Inferring luminescent chemical reactions

Determining the rates of formation in chemical reactions is a critical challenge in both biology and chemistry. The difficulty in discovering the underlying structure of chemical reactions is that there are often intermediate reactions from compounds that are neither the reactants (inputs), or products (outputs). These hidden variables are a common feature of chemical reactions and the ability to infer their existence from observing just the dynamic behavior of the outputs is an important problem for automated science.

Currently, reaction rates are determined by first principles and tedious experimental work. However, this process is difficult to apply to complex reactions with limited prior knowledge and unknown numbers of intermediate components. This project is focused on designing an automated telescience approach to infer the rate of formation in luminescent chemical reactions. Two different chemical reactions were analyzed: luciferase, the primary enzyme used in bioluminescence; and Bis-2,4,6-(trichlorophenyl)oxalate, or TCPO, the primary chemical in the complex reaction in glowsticks.

This work presents results of high-fidelity simulations of a mixing chamber and photometer instrument. Using the simulation results, a machine learning algorithm inferred

multiple rates of reactions which resemble known models of simpler systems, namely the Michaelis-Menten kinetics.

This section begins with a brief overview that describes the Michaelis-Menten kinetics as well as how the luminescent reactions approximate this behavior. The active learning framework is described in Section 5.2.2 and divided into three subsections: experimental design, experimental device, and model inference. The results of the luciferase and TCPO experiments are summarized in Section 5.2.3 and Section 5.2.4, respectively.

5.2.1 Background

This section presents an overview of the background chemistry required to understand the luminescent chemical reactions.

Michaelis-Menten kinetics

The Michaelis-Menten kinetics is one of the simplest and best-known models of enzyme kinetics [131]. An enzyme is a large biological molecule that acts as a catalyst that greatly accelerates both the rate and specificity of metabolic reactions. Enzyme kinetics is the study of the reaction rate of these enzymes and how varying conditions affect the reaction.

The Michaelis-Menten kinetics is a simple reaction that involves an enzyme, E , binding to a substrate, S , to form an enzyme-substrate complex, ES , which in turn is converted into a product, P , and the enzyme. This is summarized into the following two

chemical reactions:



where the square brackets $[\cdot]$ represents the concentration of a chemical compound, k_1 , k_2 and k_3 are the rate constants and Eq. 5.1 is a reversible process.

From Eqs. 5.1-5.2, the rate equations can be determined for each component:

$$\frac{d[E]}{dt} = -k_1[E][S] + k_2[ES] + k_3[ES] \quad (5.3)$$

$$\frac{d[S]}{dt} = -k_1[E][S] + k_2[ES] \quad (5.4)$$

$$\frac{d[ES]}{dt} = k_1[E][S] - k_2[ES] - k_3[ES] \quad (5.5)$$

$$\frac{d[P]}{dt} = k_3[ES] \quad (5.6)$$

The Michaelis-Menten kinetics depends on three assumptions:

1. **Concentration assumptions** – The concentration of the product is negligible compared to the substrate and the concentration of the substrate is significantly greater than the enzyme. Thus, reactions are continuously occurring and analysis of individual interactions is not required.
2. **Static enzyme concentration** – The total amount of enzyme remains constant during the course of the reaction. As a result, the sum of the concentrations of the free enzyme and the enzyme-substrate complex is constant, denoted by $[E_0]$ or the initial amount of enzyme:

$$[E_0] = [E] + [ES] \quad (5.7)$$

3. **Steady-state approximation** – Since the concentration of substrate is significantly larger than the concentration of enzyme and reactions are continuously occurring, the reaction is considered to be in steady state where the enzyme-substrate complex is being formed and consumed at the same rate:

$$\frac{d[ES]}{dt} = 0 \quad (5.8)$$

Combining Eq. 5.5 with Eq. 5.8 obtains:

$$0 = k_1[E][S] - (k_2 + k_3)[ES] \quad (5.9)$$

Applying the static enzyme concentration assumption, Eq. 5.7, to Eq. 5.9 obtains:

$$0 = k_1([E_0] - [ES])[S] - (k_2 + k_3)[ES] \quad (5.10)$$

$$0 = k_1[E_0][S] - (k_1[S] + k_2 + k_3)[ES] \quad (5.11)$$

$$[ES] = \frac{[E_0][S]}{\frac{k_2+k_3}{k_1} + [S]} \quad (5.12)$$

Letting $k_M = \frac{k_2+k_3}{k_1}$ and substituting Eq. 5.12 into Eq. 5.6 results in the equation for Michaelis-Menten kinetics:

$$\frac{d[P]}{dt} = \frac{k_3[E_0][S]}{k_M + [S]} \quad (5.13)$$

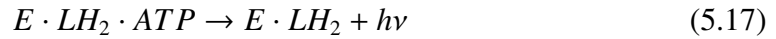
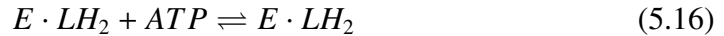
Luciferase biochemical reaction

The luciferase enzyme is the biochemical compound responsible for bioluminescence. Although the reaction is a multi-step process, it is nonetheless well approximated by Michaelis-Menten kinetic and follows the form:



where E is luciferase (the enzyme), LH_2 is luciferin, ATP is adenosine triphosphate (the substrate), PP_i are inorganic pyrophosphates, L is dehydroluciferin, AMP is adenosine monophosphate and $h\nu$ are photons (the product).

Despite the complexity of this reaction, it can be greatly simplified by mixing luciferin and luciferase together prior to the addition of ATP , which effectively forms a single enzymatic complex due to their high affinity for each other. This allows for the conceptually simplified model of:



By treating $E \cdot LH_2$ as the enzyme, ATP as the substrate and $h\nu$ as the product, the Michaelis-Menten kinetics for this reactions are:

$$\left. \frac{d(h\nu)}{dt} \right|_0 = \frac{k_1[E \cdot LH_2][ATP]}{k_M + [ATP]} \quad (5.18)$$

For simplicity and legibility, the biochemical compounds are aliased using single character symbols as described in Table 5.1. Thus, Eq. 5.18 can be rewritten as:

$$v = \frac{k_1[f][l][a]}{k_M + [a]} \quad (5.19)$$

5.2.2 Active learning framework

The automated science uses an active learning framework that consists of a series of modularized processes, which communicate through predefined methods. The framework follows the architecture described in Section 2.2. The overall architecture is summarized in the flowchart in Fig. 5.2.

Table 5.1: A list of simplified symbols for the luciferase reaction.

Original symbol	Simplified symbol
$\frac{d(h\nu)}{dt}\big _0$	ν
E	f
LH_2	l
ATP	a

Evolve experiment

The first process is responsible for designing experiments, which uses genetic algorithms to evolve experiments according to the constraints defined in a experiment configuration file. The processes begin with experiment parameters randomly drawn from a uniform distribution within the specified bounds. For the remaining iterations, the algorithm finds experiment parameters that create the largest disagreement between the proposed models from the **Evolve models** subsection. This is measured by the variance in the predicted outputs of each model. Each experiment is defined by a list of experiment parameters, stored in a comma-separated value, plain-text file (Fig. 5.3).

Physical experiment

The second process is responsible for running the experiments defined in the **Evolve experiment** subsection and recording the corresponding data. A high-fidelity MATLAB simulation of the underlying Luciferase reactions is executed and processed. The reaction rates are calculated and stored in a comma-separated value, plain-text file (Fig. 5.4).

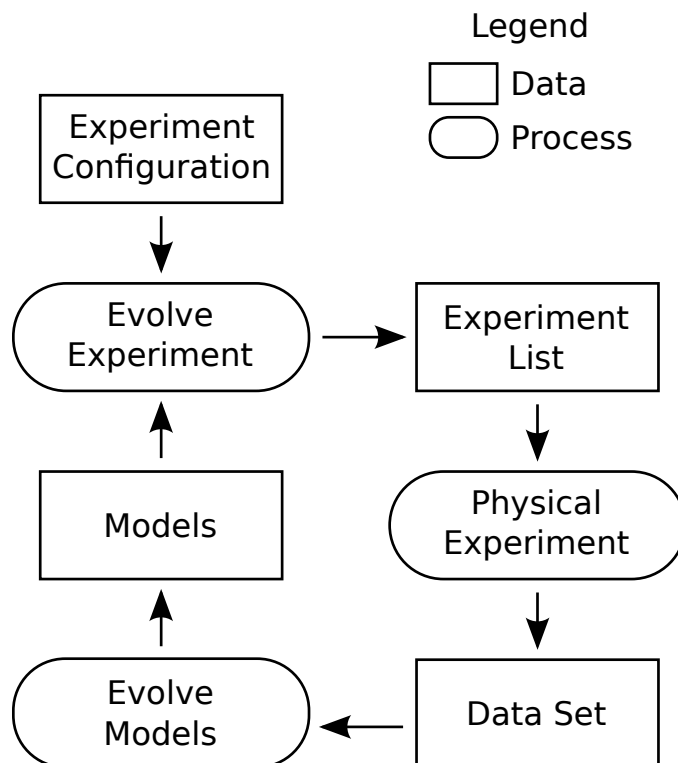


Figure 5.2: A flowchart of the active learning framework for inferring luciferase bioluminescent reactions.

```
#f concentration, l concentration, a concentration
19,100,8738771
71,100,1056984
```

Figure 5.3: An example of two experiments specified in a plain-text file. The initial concentrations of each input is defined.

Evolve models

The third process is responsible for inferring models given the results of the physical experiments. It uses genetic programs to evolve models that model the relationship between the experiment parameters and results from the **Physical experiment** subsection. It reads in the data set, applies symbolic regression, and saves the best models in the

```
#v, f, l, a
0.032197,20,100,7364519
0.038961,89,100,1273346
```

Figure 5.4: An example of the results of two experiments logged in a plain-text file. The output, v , is a function of the inputs, $\{f, l, a\}$.

model file. The symbolic regression engine is executed via the Eureqa API [177]. For additional details, refer to Section 1.3.2. Each model is described by a symbolic expression stored in a plain-text file (Fig. 5.5).

```
v = 0.0013*f + 6.006e-10*a
v = 1.39*f*a/(2002361216 + 615*f)
```

Figure 5.5: An example of two inferred models of luciferase enzyme kinetics in a plain-text file.

5.2.3 Luciferase reaction model

Three experiment series were executed to determine the viability of the infrastructure as well as the subprocesses.

Series 1 – Luciferase and ATP

The first series was configured to determine the relationship between luciferase and ATP. The concentration of the substrate ATP was allowed to vary from 0 to 10,000,000. For the enzymes, the luciferin concentration was held at a constant value of 100, while luciferase concentration was allowed to range from 0 to 80. This ensured that luciferase

would be the limiting factor in the formation of the enzyme complex and, thus, the enzyme concentration is trivially determined by the luciferase concentration.

The complete experiment loop was executed ten times, resulting in a total of 80 physical experiments. The model was able to find models that were very similar in form to Michaelis-Menten dynamics as early as the second iteration.

In each iteration, the process **Evolve experiment** was allowed to evolve for 2000 generations resulting in 8 experiments. A scatter plot of the complete set of experiment parameters is shown in Fig. 5.6. The experiment parameters did not follow a uniform sampling grid but rather focused on specific regions of interest – in particular, the boundary constraints were extremely useful in differentiating models.

The **evolve models** was allowed to evolve for 50,000 generations and was provided with the basic arithmetic building blocks (+, −, ×, /).

The final proposed model was:

$$v = \frac{0.04634fa}{59396324 + 21.4630a} \quad (5.20)$$

With some algebraic manipulations, this model predicts the Michaelis-Menten kinetics form exactly, with the rate constants as $k_3 = 0.002159$ and $k_M = 2.767 \times 10^6$.

Series 2 – Luciferin and ATP

The second series was configured to be similar to the first experiment, but instead determined the relationship between luciferin and ATP, as luciferase concentration was held at a constant value of 100, while luciferin concentration ranged from 0 to 80.

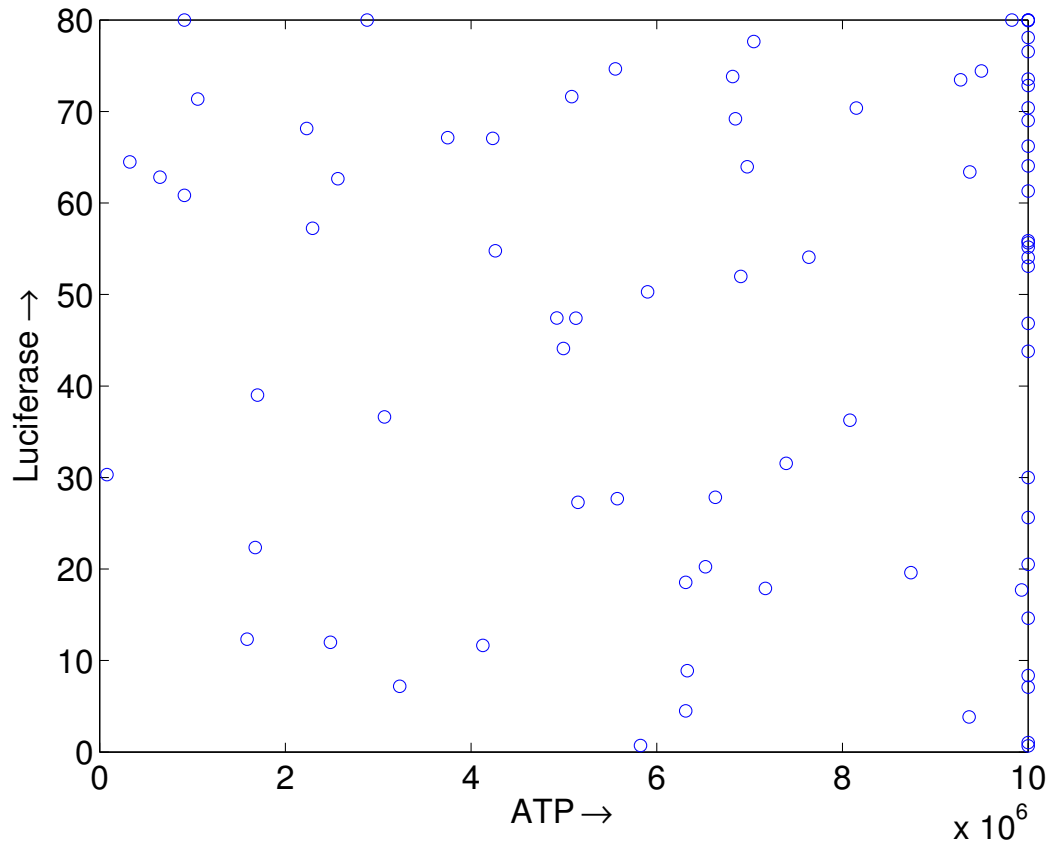


Figure 5.6: Experiment parameters for luciferase and ATP.

Unsurprisingly, the experiment parameters were evolved with a similar distribution to the first experiment. A scatter plot of the complete set of experiment parameters is shown in Fig. 5.7.

The final proposed model was:

$$v = \frac{0.002172fa}{2883780 + a} \quad (5.21)$$

Again, this model predicts the Michaelis-Menten kinetics form exactly, with consistent rate constants ($k_3 = 0.002172$ and $k_M = 2.884 \times 10^6$).

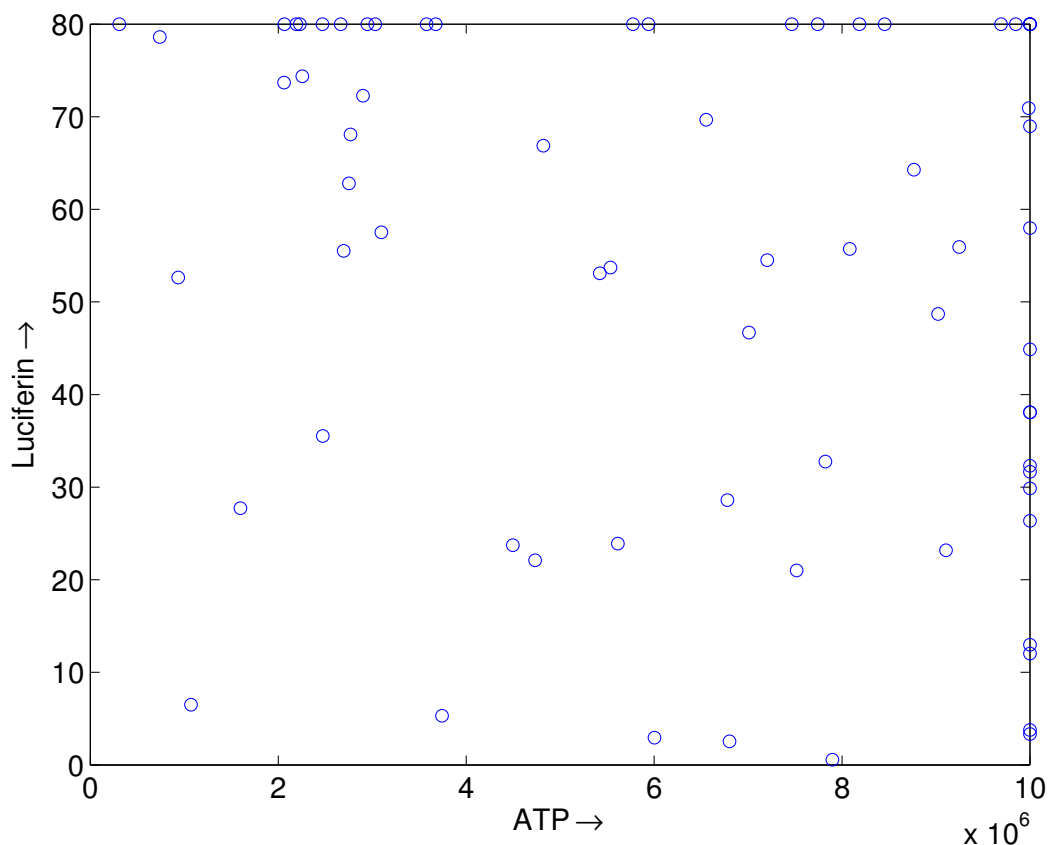


Figure 5.7: Experiment parameters for luciferin and ATP.

Series 3 – Luciferase, Luciferin and ATP

The final series provided the system access to all three inputs: luciferase, luciferin and ATP, with concentration bounds of (0, 500), (0, 500), (0, 10000000), respectively. This series was to determine whether the learning algorithm would be able to infer that luciferase and luciferin are limiting factors in addition to the Michaelis-Menten kinetics.

The complete experiment loop was 64 times. The process **Evolve experiment** generation limit was kept at 2000, while the evolve models was extended to 100,000 generations.

The learning algorithm was able to find a variety of expressions that appear to be viable kinetic models:

$$v_0 = \frac{(2.2344 \times 10^{-5})[E][LH2][ATP]}{3186854 + [ATP] + 0.1092[E][LH2]} \quad (5.22)$$

$$v_0 = \frac{(2.2349 \times 10^{-5})[E][LH2][ATP]}{3177640 + [ATP] + 78.6592[LH2]} \quad (5.23)$$

$$v_0 = \frac{(2.2323 \times 10^{-5})[E][LH2][ATP]}{3200075 + [ATP]} \quad (5.24)$$

Although none of these equations are identical to the two-substrate Michaelis-Menten models (Eq. 5.19), they appear to be legitimate equations that could be derived from rate equations. Perhaps the difference in these equations arise when the Michaelis-Menten assumptions no longer hold (namely Eqs. 5.7-5.8) and additional analysis of these kinetic models is left for future work.

5.2.4 TCPO reaction model

This section focuses on inferring the chemical reaction of Bis-2,4,6-(trichlorophenyl)oxalate, or TCPO, the primary chemical in glowsticks. TCPO was chosen as a subject since its reaction is easier to reproduce with an automated instrument in a controlled environment. This work presents results of a high-fidelity, multiphysics simulation to gather synthetic data of a mixing chamber and photometer instrument.

Device

Due to physical experimental constraints, the luciferase reaction was abandoned in favor of the TCPO reaction. A physical device was constructed to automate the testing of the

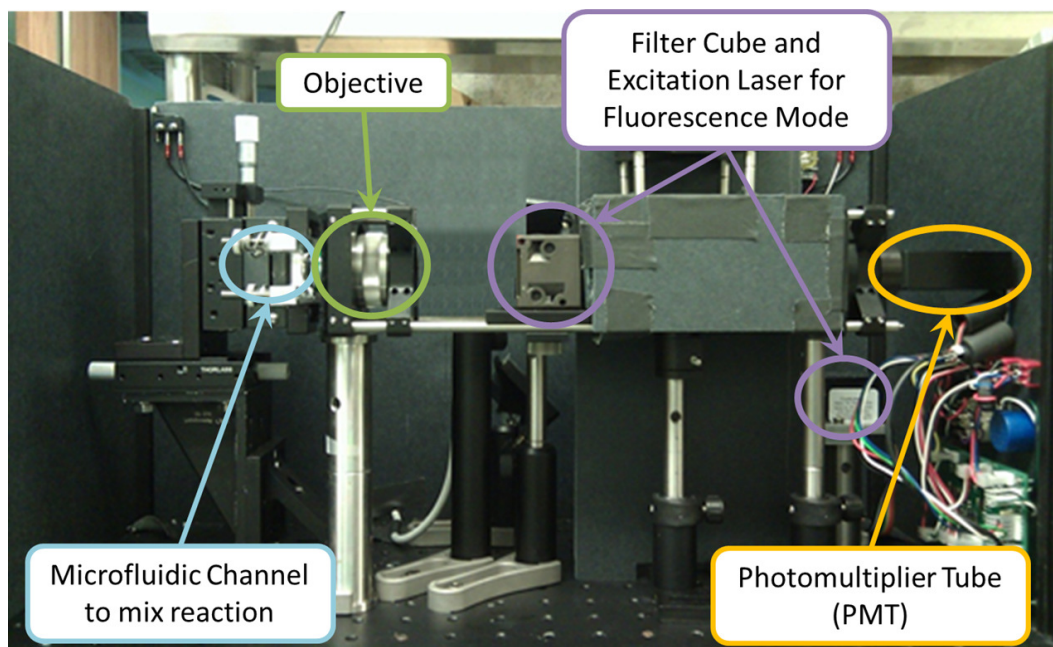


Figure 5.8: The luminescent chemistry instrument for inferring the TCPO reaction. The primary components of the device are labeled. Image courtesy of Christina Marasco.

TCPO reaction, which consists of a microfluidic mixing channel and a photomultiplier tube sensor (Fig. 5.8).

Furthermore, initial experimental work used a high-fidelity, multiphysics simulation to produce accurate predictions of the physical device. The microfluidic chamber was simulated in COMSOL and accounted for the spatial and temporal dynamics of the reaction, compared to simulating just the reaction rates for the luciferase reaction. An image of a simulation from the microfluidic chamber is shown in Figure 5.9.

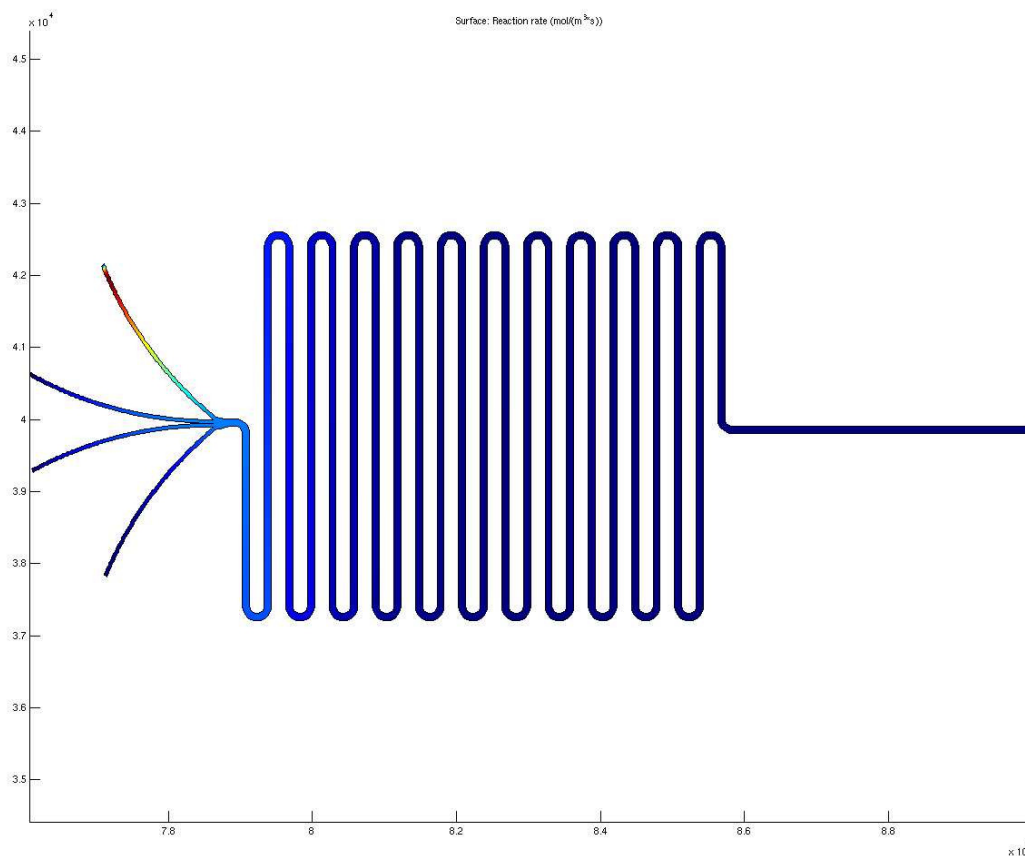
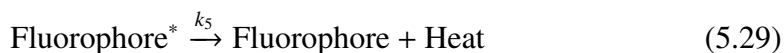
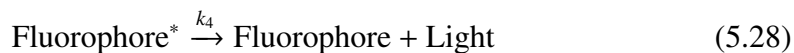
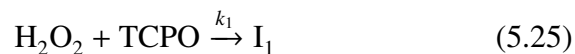


Figure 5.9: A COMSOL multiphysics simulation of the microfluidic mixing chamber for the TCPO reaction. Image courtesy of Christina Marasco.

TCPO reaction

The TCPO reaction is a complex reaction with many different chemical compounds [36]. It consists of three inputs: hydrogen peroxide (H_2O_2) and Bis-2,4,6-(trichlorophenyl)oxalate (TCPO) are the two substrates while Fluorophore is the enzyme. The products of the reaction are heat and light. The reaction also includes four intermediate compounds, which are labeled as I_n , respectively: trichlorophenol (I_1), an unstable peroxyacid ester (I_2), phenol (I_3) and 1,2-dioxetanedione (I_4).

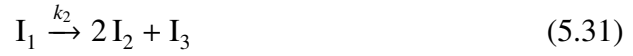


For the sake of clarity, the nomenclature has been simplified according to their roles in Table 5.2.

Table 5.2: The simplified nomenclature for the TCPO reaction according to the compound's role.

Role	Name	Old symbol	New symbol
Substrates	H_2O_2	H_2O_2	S_1
	TCPO	TCPO	S_2
Enzyme	Fluorophore	F	E
Enzyme-Substrate Complex	Fluorophore*	Fluorophore*	ES
Products	Light	Light	P_1
	Heat	Heat	P_1
Intermediates	trichlorophenol	I_1	I_1
	unstable peroxyacid ester	I_2	I_2
	phenol	I_3	I_3
	1,2-dioxetanedione	I_4	I_4

Using the naming convention in Table 5.2, the Eqs. (5.25)–(5.29) can be rewritten as:



The rate equations can be determined for each component:

$$\frac{d[S_1]}{dt} = -k_1[S_1][S_2] \quad (5.35)$$

$$\frac{d[S_2]}{dt} = -k_1[S_1][S_2] \quad (5.36)$$

$$\frac{d[I_1]}{dt} = k_1[S_1][S_2] - k_2[I_1] \quad (5.37)$$

$$\frac{d[I_2]}{dt} = k_2[I_1]^{\frac{1}{2}} \quad (5.38)$$

$$\frac{d[I_3]}{dt} = k_2[I_1] + k_6[ES][I_4]^2 - k_3[I_3][E] \quad (5.39)$$

$$\frac{d[I_4]}{dt} = k_3[I_3][E] - k_6[ES][I_4]^2 \quad (5.40)$$

$$\frac{d[E]}{dt} = k_6[ES][I_4]^2 + k_4[ES] + k_5[ES] - k_3[I_3][E] \quad (5.41)$$

$$\frac{d[ES]}{dt} = k_3[I_3][E] - k_6[ES][I_4]^2 - k_4[ES] - k_5[ES] \quad (5.42)$$

$$\frac{d[P_1]}{dt} = k_4[ES] \quad (5.43)$$

$$\frac{d[P_2]}{dt} = k_5[ES] \quad (5.44)$$

Inferred equations

Using the same infrastructure described in Section 5.2.3, the experiment was configured to determine the relationship between H₂O₂, TCPO and Fluorophore. The flow concen-

trations for H_2O_2 , TCPO and Fluorophore were $9.79[\text{mol}/\text{L}]$, $0.011138413[\text{mol}/\text{L}]$, and $0.000563169[\text{mol}/\text{L}]$, respectively. Each compound was added as a channel in the microfluidic chamber and a fourth channel of passive fluid was added to system to ensure the total flow in the chamber was constrained to $10^{-10}[\text{m}^3/\text{s}]$.

The complete experiment loop was executed 28 times, including 3 initial random experiments. In each iteration, the process **Evolve experiment** was allowed to evolve for 5000 generations and the process **Evolve model** was allowed to evolve for 10^9 evaluations.

The inferred equation for the light production of TCPO was:

$$\frac{d[P_1]}{dt} = \frac{k_\alpha[S_1]^2[S_2]^2}{k_\beta + [S_2]^2} \quad (5.45)$$

A primary constraint that was applied during all the experiments was that the Fluorophore and TCPO were dispensed from the same pump ($[E] = \lambda[S_1]$). Because the equation is a fractional expression, components can be canceled out algebraically, which ultimately could hide some of the key dynamics. For example, both of the following equations are plausible expressions that would resolve to equation (5.45):

$$\frac{d[P_1]}{dt} = \frac{k_\alpha[E][S_1][S_2]^2}{k_\beta + [S_2]^2} \quad (5.46)$$

$$\frac{d[P_1]}{dt} = \frac{k_\alpha[E]^2[S_1]^2[S_2]^2}{k_\beta[E]^2 + [S_1]^2[S_2]^2} \quad (5.47)$$

Follow the Michaelis-Menten derivation in Section 5.2.1 and assuming Eq.(5.47) to be true, the following relationships can be obtained:

$$(5.47) \rightarrow \frac{d[P_1]}{dt} = \frac{k_{\alpha}[E]^2[S_1]^2[S_2]^2}{k_{\beta}[E]^2 + [S_1]^2[S_2]^2} \quad (5.48)$$

$$(5.48) + (5.43) \rightarrow [ES] = \frac{k_{\alpha_1}[E]^2[S_1]^2[S_2]^2}{k_{\beta}[E]^2 + [S_1]^2[S_2]^2} \quad (5.49)$$

$$(5.49) \rightarrow 0 = k_{\alpha_1}[E]^2[S_1]^2[S_2]^2 - k_{\beta}[ES][E]^2 - [ES][S_1]^2[S_2]^2 \quad (5.50)$$

5.2.5 Conclusions and future work

A basic infrastructure was presented that is capable of automatically inferring Michaelis-Menten kinetics from the luciferase and TCPO luminescent reactions by iteratively designing experiments, running experiments and building models. The entire system operates without user interaction except for setting the initial limits on the experiment configuration. The infrastructure operates locally on a single machine and passes information through comma-separated value, plain-text files. Future work includes additional testing and analysis on the hidden compounds in the inferred expressions.

5.3 Inferring multi-compartment pharmacokinetics models

Pharmacokinetics is the study of the rates of absorption, distribution, metabolism and excretion of administered drugs in a living organism [99]. The accuracy of pharmacokinetic models is integral to drug design as drugs must be administered such that their concentrations become neither too low that they are neither ineffective nor too high that they become toxic. However, pharmacokinetic models are difficult to infer, as they are complex dynamical systems.

Furthermore, the model inference problem is further complicated by the need for multi-compartmental models which presents a heterogeneous inference challenge. Compartments are defined volumes of body fluids and have unique heterogeneous dynamics [159]. Example of body compartments include blood plasma, interstitial fluid, fat tissue, intracellular fluid and transcellular fluid. Blood plasma has a low volume with high distribution rates while fat tissue has a relatively high volume that is capable of storing drug concentrations. The rates and dynamics of each compartment varies by drug, which adds to the challenge of inferring multi-compartment pharmacokinetics models.

Multi-compartment pharmacokinetics is an ideal field for automated telescience. It allows for the construction of repeatable experiments with rich time-series measurements of drug concentrations. The ability to actively probe the system with varying drug concentrations is ideal for the automated modeling of complex, nonlinear phenomena.

As a first step towards inferring multi-compartment pharmacokinetic models using remote controlled devices, a microfluidic analogy was built in collaboration with the Vanderbilt Institute of Integrative Biosystems Research and Education at Vanderbilt University led by Professor John P. Wikswo. The device pumped dyes of varying opacities and the concentration of the dyes was measured at various wells using a microscope. This framework was a suitable analogy to multi-compartment pharmacokinetics models and it is also easily extendable to more complex phenomena using microfluidic chambers with biochemical reactions.

This section begins with a discussion of the physical device in Section 5.3.1, followed by an overview of the experiment design and model inference of discontinuous differential equations in Section 5.3.2 and 5.3.3, respectively. Initial experiments on a synthetic testbench are presented in Section 5.3.4 and the results of remote operated ex-

periments on the physical device is presented in Section 5.3.5. The section is concluded in Section 5.3.6.

5.3.1 Physical multi-compartment, microfluidic device

The physical device for the automated discovery of multi-compartment microfluidic chambers consists of three remote controlled pumps with syringes filled preloaded dyes. The pumps inject the fluid into a custom microfluidic chamber that is situated on the base of a remote controlled microscope, which is able to record the transmittance of light at predefined locations on the device. The device is shown in Figs. 5.10 and 5.11.



Figure 5.10: The initial setup of the physical, multi-compartment microfluidic device. The microfluidic device filled with dye is shown on the left, while the pump-microscope-computer assembly is shown on the right. Image courtesy of Philip Samson.

It is important to note that the layout of the microfluidic chamber itself has never been revealed and that this research project followed a blind experiment protocol. The microfluidic device was designed and set up independently by the collaborators in the Vanderbilt Institute of Integrative Biosystems Research and Education.



Figure 5.11: The final setup of the physical, multi-compartment microfluidic device. Image courtesy of Philip Samson.

The pumps and microscope are controlled by a local computer. The computer hosts an FTP server, which continuously polls an upload directory for new commands. The command file is moved to an archive directory and processed for execution. After running the experiment and recording the measurements, the resulting output file is dumped to a results directory for user retrieval. All the files are comma-separated value, plain text files for ease of parsing and analysis.

Command list

Each experiment is defined using a list of commands or instructions. The commands are processed in a serial order using a blocking structure that does not proceed until the previous command is completed. Each command is referenced using a command ID and certain commands take an operand or argument.

Table 5.3: The list of commands for device control.

Command ID	Command Label	Number of operands	Bounds
1	Move to home	0	-
2	Sleep in seconds	1	$[0, \infty)$
3	Sleep in microseconds	1	$[0, \infty)$
4	Get data value	0	-
5	Pump 1 on	0	-
6	Pump 2 on	0	-
7	Pump 3 on	0	-
8	Set pump 1 rate [%]	1	$[0, 100]$
9	Set pump 2 rate [%]	1	$[0, 100]$
10	Set pump 3 rate [%]	1	$[0, 100]$
11	Pump 1 off	0	-
12	Pump 2 off	0	-
13	Pump 3 off	0	-
14	Move to data point 1	0	-
15	Move to data point 2	0	-
16	Move to data point 3	0	-
17	Move to data point 4	0	-

The list of commands is described in Table 5.3. The commands can be roughly divided into three groups: moving the microscope device (commands 1, 14-17), halting for a prespecified amount of time (commands 2-3), recording the microscope value, and controlling the pumps (commands 5-13).

Due to the serial nature of the command processing and instruction set, it is important to note that only a single physical location can be recorded at a time. If the user

wishes to retrieve the information about two locations, the microscope must record a single point first, move to the second location, and then record the second point. Thus, it is impossible to obtain synchronized recordings from all the data points simultaneously.

Command file and results example

Figure 5.12 depicts an example of a comma-separated value command file that is ready for processing and execution. The commands in Table 5.3 are organized into a sequence of instructions that describe the intended experiment.

```
#instruction number, command id, operand
1,8,0      #set pump 1 rate to 0
2,9,0      #set pump 2 rate to 0
3,10,100   #set pump 3 rate to 100
4,11       #turn pump 1 off
5,12       #turn pump 2 off
6,7        #turn pump 3 on
7,2,270    #wait 270s
8,14       #move to data point 1
9,2,2      #wait 2s
10,4       #record data from data point 1
11,15      #move to data point 2
12,2,2     #wait 2s
13,4       #record data from data point 2
14,16      #move to data point 3
15,2,2     #wait 2s
16,4       #record data from data point 3
17,17      #move to data point 4
18,2,2     #wait 2s
19,4       #record data from data point 4
```

Figure 5.12: An example of an executable command file. Note that comments are denoted by # and are only embedded in this example for ease of interpretation.

Figure 5.13 shows the corresponding results file from the command file in Fig. 5.12. Each line in the results file corresponds directly to the same instruction number in the command file. After the processing of the command file is complete, it is uploaded as a single comma-separated value file in the results directory.

The data from the results file, Fig. 5.13, can then be extracted to generate time-series data. An example of the time-series data is shown in Fig. 5.14. Note that the data points are not synchronized and further post processing is required for model inference, which is described in Section 5.3.3.

```

Rel. Start Time[ms],Position,Command ID,Operands,Data Value,Text File
0,1,8,0,<n/a>,<n/a>,
28,2,9,0,<n/a>,<n/a>,
55,3,10,100,<n/a>,<n/a>,
81,4,11,<n/a>,<n/a>,<n/a>,
86,5,12,<n/a>,<n/a>,<n/a>,
91,6,7,<n/a>,<n/a>,<n/a>,
96,7,2,270,<n/a>,<n/a>,
270098,8,14,<n/a>,<n/a>,<n/a>,
270099,9,2,2,<n/a>,<n/a>,
272100,10,4,<n/a>,0.024414,/home/cornell/results/text/10_data.txt,
272180,11,15,<n/a>,<n/a>,<n/a>,
272180,12,2,2,<n/a>,<n/a>,
274182,13,4,<n/a>,0,/home/cornell/results/text/13_data.txt,
274248,14,16,<n/a>,<n/a>,<n/a>,
274248,15,2,2,<n/a>,<n/a>,
276250,16,4,<n/a>,0.195312,/home/cornell/results/text/16_data.txt,
276314,17,17,<n/a>,<n/a>,<n/a>,
276314,18,2,2,<n/a>,<n/a>,
278316,19,4,<n/a>,0,/home/cornell/results/text/19_data.txt,

```

Figure 5.13: An example of a results file generated from the command file in Fig. 5.12.

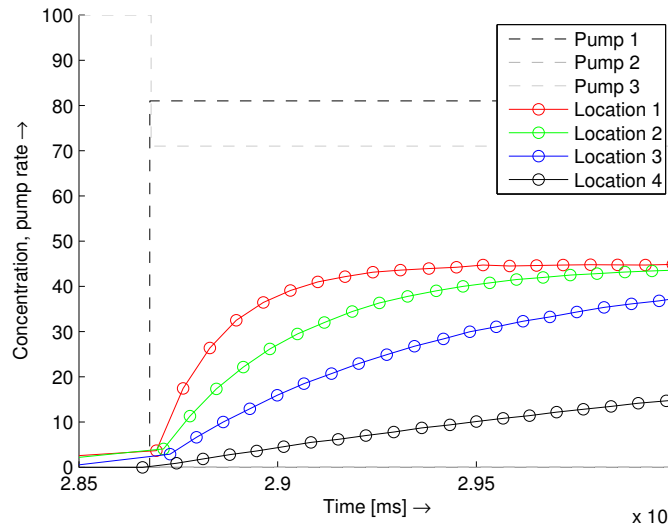


Figure 5.14: An example of the time-series data extracted from a results file.

5.3.2 Experiment design

This section describes how experiments are parameterized and designed using active machine learning algorithms.

Experiment parameterization

Each pump is represented as a square wave and described completely by four independent parameters:

1. **Pump on rate** – this parameter defines the pump rate when it is turned on.
2. **Pump on cycle** – this parameter defines the length of the period when the pump is turned on.
3. **Pump off cycle** – this parameter defines the length of the period when the pump is turned off.
4. **Pump offset** – this parameter defines a phase shift in the square wave.

When the pump is turned off, the pump rate is set to zero and the command to turn the pump off is explicitly called (command IDs 11-13 in Table 5.3). The pump offset is important as it allows pumps to operate independently from each other and avoid undesired synchronization, which can mask the individual contributions of each pump the dynamics of the device.

Each parameter has an upper and lower limit, described in Table 5.4. The limits of the parameters were chosen in response to the dynamics of the device. The pump on rate was chosen to exploit the full range of the capabilities of the pump. The lower limit

Table 5.4: The upper and lower limits of each pump parameter.

Pump parameter	Lower limit	Upper limit	Unit
Pump on rate	0	100	%
Pump on cycle	100	300	number of atomic measurements
Pump off cycle	100	300	number of atomic measurements
Pump offset	0	400	number of atomic measurements

of the cycle parameters was chosen to ensure that the dynamics would reach steady-state behavior, an issue addressed in Section 5.3.3, while the upper limit of the cycle parameters was chosen to ensure the system would not remain in steady-state for the majority of an experiment and contain a significant portion of redundant information. The pump offset was chosen to ensure that each pump would be able to begin in both an on and off state. The parameters of the square waves are depicted in Fig. 5.15.

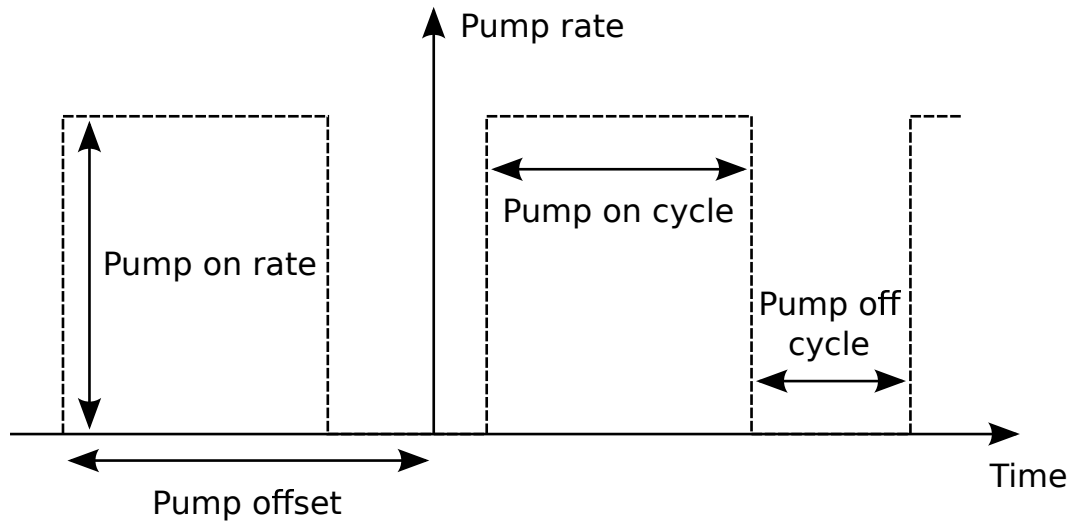


Figure 5.15: A diagram of the square wave defined by the corresponding pump parameters.

Generating command files from experiment parameters

This section describes how command files are generated from experiment parameters.

The process is divided into four different subprocedures:

1. **Initialization procedure** – this procedure ensures that the experiment begins at a known initial state, namely all the location are guaranteed to have zero concentration. This is critical for dynamic systems since the evolution of the system is dependent on the initial conditions, and consistent initial conditions ensures both consistent results as well as predictable behavior required for experiment design.

The initialization procedure begins by turning pumps 1 and 2 off and turning pump 3 on at 100% pump rate for 270 seconds. The settings in this procedure was determined manually through trial and error, and was shown to consistently ensure the concentrations were zero while requiring the minimal amount of time. The pump rates are all turned off, and this marks the beginning of the active experiment.

2. **Experiment execution** – this procedure executes the experiment defined by the square wave parameters.

An important aspect in the generation of square waves is the concept of atomic measurements. Since commands can only be executed in a serial fashion, it is impossible to read all the locations simultaneously with the existing infrastructure. As a result, an indivisible unit of measurement is created, which is guaranteed to read all the locations in order. The microscope is instructed to move to the first data point and wait for 100ms before reading the concentration at that location. This set of three commands is repeated for the remaining three locations, and the set of twelve instructions constitutes a single atomic measurement. The 100ms

delay is included to ensure that the device is stable and any effects produced by moving the microscope are negligible.

This procedure commits one set of atomic measurements, and then checks the state of each pump to determine whether the pump needs to be turned on or off depending on their independent periods. This continues until the experiment executes for 900 atomic measurements, which was guaranteed to contain at least 1.5 cycles and at most 4.5 cycles of the pump square wave. This active experiment component requires approximately 625 seconds of clock time.

3. **Shutdown procedure** – this procedure ensures that the device is in working order and requires minimal maintenance. Microfluidic devices are capable of capturing air bubbles in their chambers, which drastically distort the intended performance of the device. As a result, all the pumps are set at a 5% flow rate to ensure constant fluid flow, which minimizes the creation of bubbles until the next experiment is executed.
4. **Experiment validation** – as described in Section 5.3.3, the post processing of the raw time-series data is done using segmented smoothing splines that fit each discontinuous section created by the square wave inputs. An unintended consequence of cubic smoothing splines is that each section must contain at least seven data points. If the experiment parameters dictates that two pumps change states less than seven atomic measurements apart, then the smoothing splines will be unable to fit this section of data. Thus, this procedure checks that all the discontinuous sections contain enough data for the smoothing spline post processing. If the experiment fails this check, then it is deemed invalid and immediately rejected.

Random experiment design

The design of random experiments is critical for a number of applications, including generating validation data sets as well as to generate an initial training set that acts as a seed for the active learning process. To generate a random experiment, each pump parameter is set to a random number drawn from a uniform distribution within the bounds in Table 5.4.

Optimal experiment design

The majority of experiments are designed using the optimal experiment design framework outlined in Section 2.2. This is an ideal application for optimal experiment design as it provides an interactive environment with a fixed modeling problem, namely for each location, find the function, f_i , that models:

$$\dot{x}_i = f_i(x_1, x_2, x_3, x_4, u_1, u_2, u_3) \quad (5.51)$$

where x_i is the concentration of the i^{th} location and u_j is the pump rate of the j^{th} pump.

The experiment design task is to find the pump rates, u_j , that maximize the surprisal in the rates of change for each location, \dot{x}_i . Although this task may appear trivial, it is significantly more complex than the simple input-output relationships described in Section 2.2. The complexity arises from the fact that the expression is an ordinary differential equation and, as a result, depends on the full state of the system, which includes the concentrations at each location, x_i . Although the experiment design can control the pumps directly, it is unable to directly control the remaining states and can only indirectly achieve the states by setting the system through a specific trajectory in its state space.

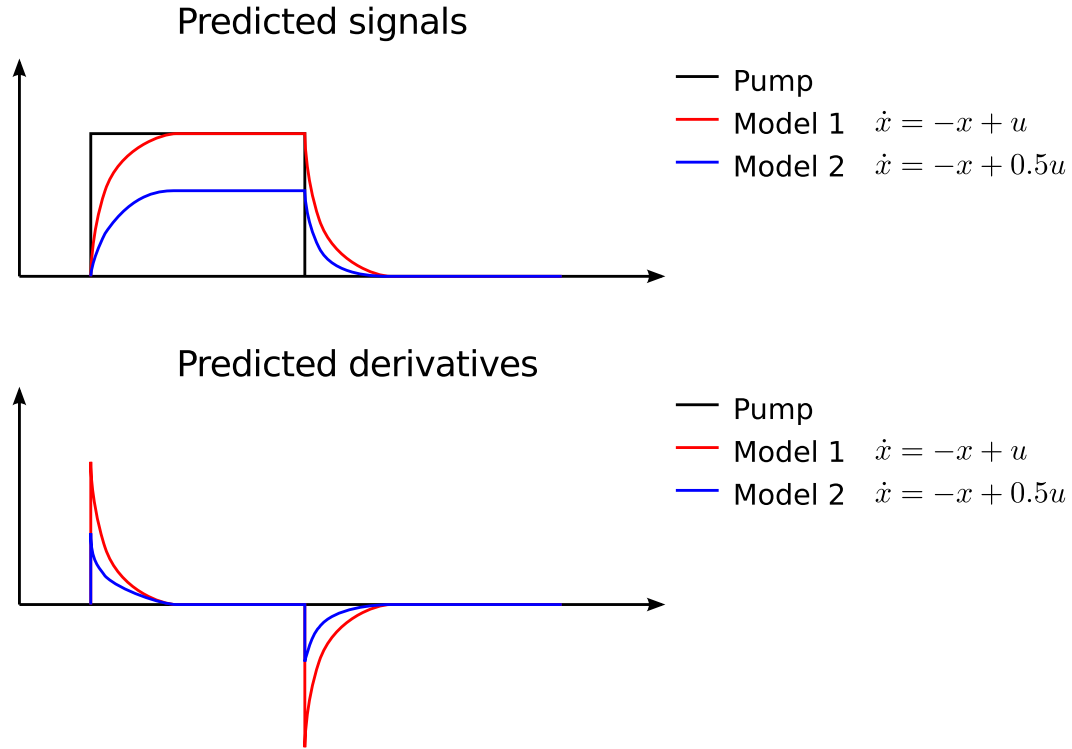


Figure 5.16: Experiment design via expected derivatives. The top plot shows the simulated signals generated by numerical integration and the bottom plot shows the derivatives of those signals. Note that the two signals predict different steady state signals, but the derivatives of those signals in the steady state region are both zero. Using expected derivatives, these two models have zero disagreement despite producing entirely different predictions.

The first approach was to design experiments by simulating the models and maximize the surprisal in expected derivatives. The problem with this approach is that different models can predict the same derivative using different states. In particular, if two models predict different steady-state conditions, the amount of surprisal or disagreement in these models is zero despite the fact that the behavior is a result of two entirely different conditions. As a result, experiments that are designed using just the predicted derivatives have a limited capacity to differentiate between models with different steady-state behavior (Fig. 5.16).

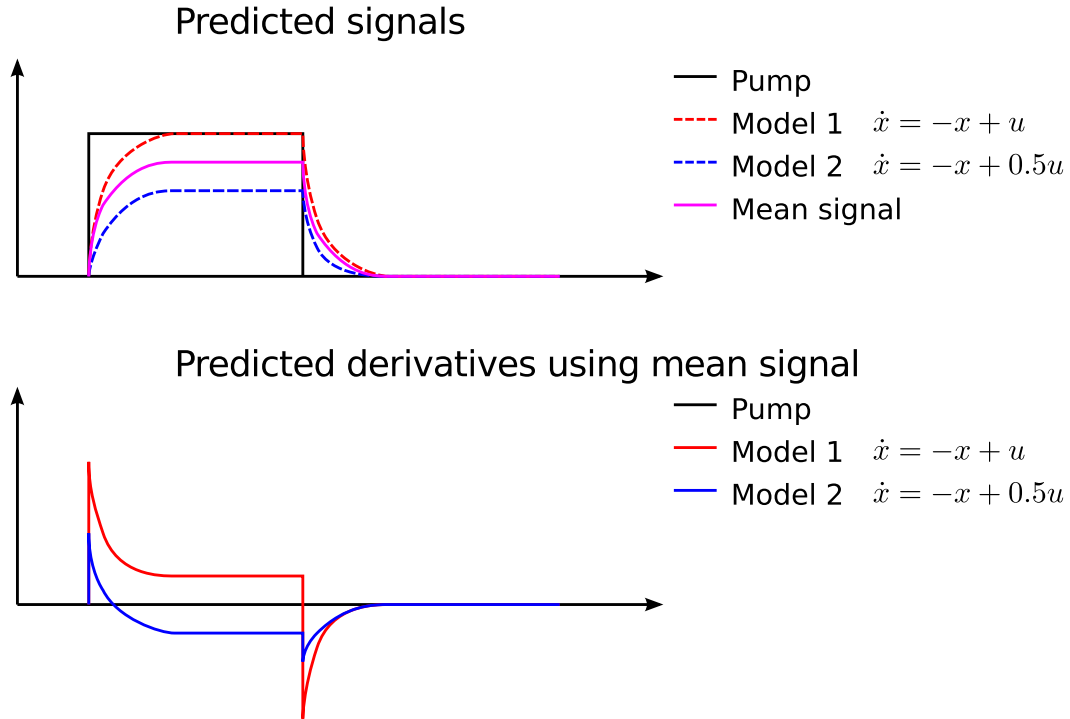


Figure 5.17: Experiment design via weighted mean signal. The top plot shows the simulated signals with the addition of the weighted mean and the bottom plot shows the predicted derivatives based on the mean signal. Note the mean signal predicts a different steady state value than the original signals and as a result, the predicted derivatives using the mean signal are different. This allows the experiment design process to find experiments that makes the models disagree on steady state values.

The second approach was to compute the weighted average of the simulated signals and use the weighted average signal to calculate the expected derivatives. Since the accuracy of each model is known in advance, the weighted mean signal is computed as a convex sum of the existing signals weighted by the empirical variance of the model (Eq. 2.29). This mean signal is then used to calculate the expected derivatives, and this approach is sensitive to different steady state predictions (Fig. 5.17).

Experiment design via managed bank

A primary issue in competitive coevolutionary algorithms occurs when one algorithm systematically outperforms the other, resulting in a loss of selection pressure for both algorithms [32, 213]. All the individuals in one population dominate the other, resulting in a loss of fitness gradient. This pathology is called disengagement [15]. For experiment design, this can occur when the experiment population predict the performance of the models to be equally accurate or poor, and the resulting models based on the new experiments do not update with any meaningful information. This effectively creates a deadlock situation for the coevolutionary system.

The approach of experiment design is based on maximizing the disagreement and information gain given candidate models. However, when the candidate models begin to show biases in a specific region, disengagement can occur. For example, maximizing model disagreement involves maximizing the variance in the derivatives, which can be achieved by producing experiments that create large derivatives. Even if the models are able to predict the derivatives relatively well, just by creating large signals amplifies any errors in the derivatives. By only focusing on large signals, this results in a loss of fitness gradient.

Instead, the ‘managed challenged’ approach, developed by Bongard and Lipson [15], is used. This approach adds a bank of experiments and applies the traditional active machine learning framework. Rather than blindly committing each experiment to the training data, it is first tested to check if it reduces the error on the validation set. If it does reduce the validation error, then the coevolutionary algorithm proceeds as normal. However, if it fails to help in generating better models, then the results are stored in a bank for future use and the algorithm begins to design experiments that aid in gen-

eralization. Once an experiment succeeds to reduce the validation error, the algorithm returns to maximizing disagreement.

The original work by Bongard and Lipson designed secondary tests by searching for experiments that produced half the fitness of the previous experiment:

$$\begin{aligned} &\text{if } x_{k-1}^* \text{ failed,} \\ &\quad x_k^* = \arg \min_x \frac{1}{2} F(x_k^*) \end{aligned} \quad (5.52)$$

where x_{k-1}^* is the previous optimal experiment, x_k^* is the current optimal experiment, and $F(x)$ is the fitness evaluation of x . This approach continually reduce the expected complexity of the problem until it could find a fitness that should be ‘digestible’ from the bank.

Rather than use the approach in Eq. 5.52, this work presents an alternative requirement for the secondary tests that emphasizes searching for experiments that are far away in the input space:

$$\begin{aligned} &\text{if } x_{k-1}^* \text{ failed,} \\ &\quad x_k^* = \arg \max_x \arg \min_n \|x_n^* - x\|^2 \end{aligned} \quad (5.53)$$

where $n \in \{1, \dots, k-1\}$ is an index that iterates over all of the previous experiments. Thus, Eq. 5.53 rewards new experiments that are in regions where little data has been previously collected. This encourages new experiments to investigate behaviors that are not currently in the training data set.

Experiment design process

The last aspect of the experiment design is the use of a repeatability experiment. Since many traditional properties are no longer guaranteed in remote experiments, new meth-

ods must be designed to deal with the loss of these properties. In local experiments, the environment are supervised and the instrument status is easily confirmed. Thus, any obvious external disruptions, such an inadvertent collisions, are identified immediately and dealt with accordingly: the data is purged and the experiment is restarted. However, with remote experiments, a robust protocol must be developed ensure that properties such as repeatability and stationarity are maintained throughout the data collection process.

Before every experiment is executed, a repeatability experiment is executed first. The repeatability experiment is a randomly generated experiment, but remains the same throughout the course of the series. Thus, any changes to the long term behavior of the device can easily be compared by referencing the repeatability experiments. Only external disruptions that occur within the time frame of a single experiment can pass undetected. Furthermore, the repeatability experiments provide a robust benchmark to compute the measurement noise of the system.

The complete experiment design process is summarized in Algorithm 5.1.

Algorithm 5.1: Experiment design for the pharmacokinetic device

```
1  #initial seeding experiments
2  for i in range(random_experiments) :
3      execute(repeatability_experiment)
4      random_experiment = generate_random()
5      execute(random_experiment)
6      add_data_to_training_set(random_experiment)
7
8  #model inference and bank initialization
9  infer_models() # (Section 5.3.3)
10 successful_design = true
11 bank.clear()
12
13 #experiment design
14 for i in range(designed_experiments) :
15     if (bank.is_empty()) :
16         if (successful_design) :
17             #generate experiments using surprisal
18             successful_design = false
19             execute(repeatability_experiment)
20             designed_experiment = generate_surprisal_experiment() # (Eq. 2.21)
21             execute(designed_experiment)
22         else :
23             #generate experiments using distance
24             execute(repeatability_experiment)
25             designed_experiment = generate_distance_experiment() # (Eq. 5.53)
26             execute(designed_experiment)
27             bank.add(designed_experiment)
28     else :
29         #attempt to commit the experiments in the bank to the training data
30         for j in range(bank.size()) :
31             add_data_temporarily_to_training_set(bank[j])
32             infer_models() # (Section 5.3.3)
33             if (reduce_validation_error())
34                 commit_data_to_training_set(bank[j]) :
35                 successful_design = true
```

5.3.3 Model inference of discontinuous differential equations

The goal of model inference is to generate first order differential equations that explains the observations at particular locations with respect to the pump rates and other locations. For each location, find the function, f_i , that models:

$$\dot{x}_i = f_i(x_1, x_2, x_3, x_4, u_1, u_2, u_3) \quad (5.54)$$

where x_i is the concentration of the i^{th} location and u_j is the pump rate of the j^{th} pump.

Smoothing splines

The experiment is executed by following a sequential list of commands, which includes defining pump rates and measuring concentrations at specific locations. Due to the sequential nature of the experiment, information is obtained in a discrete, asynchronous manner – in particular, it is impossible to measure the concentration at all the pumps simultaneously. The vast majority of machine learning algorithms, including symbolic regression, requires information to be synchronized.

As a result, smoothing splines were used to postprocess the raw time-series data. A cubic spline with a term that penalizes curvature is fit to the raw, asynchronous data via k -fold cross validation [41]. This approach provides numerous benefits:

1. Interpolation and synchronization of asynchronous data
2. Providing accurate and smooth calculations of derivatives
3. Noise rejection

Unfortunately, the square wave structure of the pump design does not generate data that is inherently suitable for smoothing splines. In particular, smoothing splines assume continuity – however, the inputs change in a discontinuous manner and this discontinuity is reflected in outputs as well. Assuming that the data is smooth and continuous introduces significant artifacts in the models (Fig. 5.18).

The initial and naive solution to this problem was to reduce the discontinuity in the pump inputs. Rather than using discontinuous square waves as inputs, a trapezoidal wave would provide a smoother input, which would result in smoother outputs. However, this approach had two underlying challenges: first, trapezoidal waves introduce greater complexity since the climbing rate presents a new parameter. Second, there are

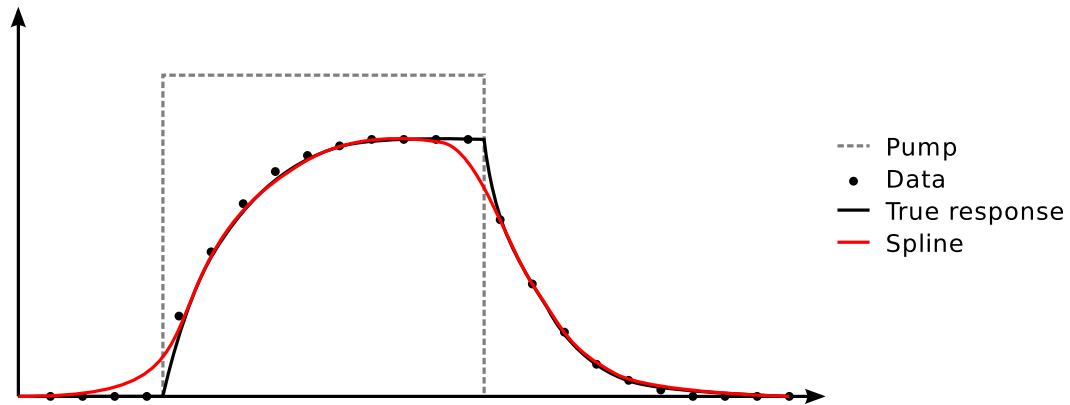


Figure 5.18: Erroneous spline fitting for discontinuous inputs. Note the difference between the spline and the true response when the pumps are turned on and off.

no experiment commands to execute continuous pump changes. Instead, the pumps are controlled by integer values set at discrete time steps. As a result, the trapezoidal wave functions are approximated by discrete steps.

While stepped trapezoidal waves seemed to be an appropriate alternative in theory, it was not effective in practice. In theory, stepped trapezoidal waves produces intricate derivatives, which critically, were not observed experimentally (Fig. 5.19). As a consequence, none of the inferred equations were dependent on the input pump rates. Thus, this approach of using stepped trapezoidal waves was inherently flawed.

To resolve the smoothing spline issue, a preprocessing step was implemented that segments the data based on discontinuities in the input and splines were fit to individual sections (Fig. 5.20). While this approach provided accurate results, it also presented a new challenge: cubic splines require a minimum number of points for fitting. Since the data is segmented when any of the three pump changes values, this approach puts a limit on how closely two pumps can change values. This challenge was resolved with some additional logic to ensure that the pumps switch sufficiently far apart and this approach

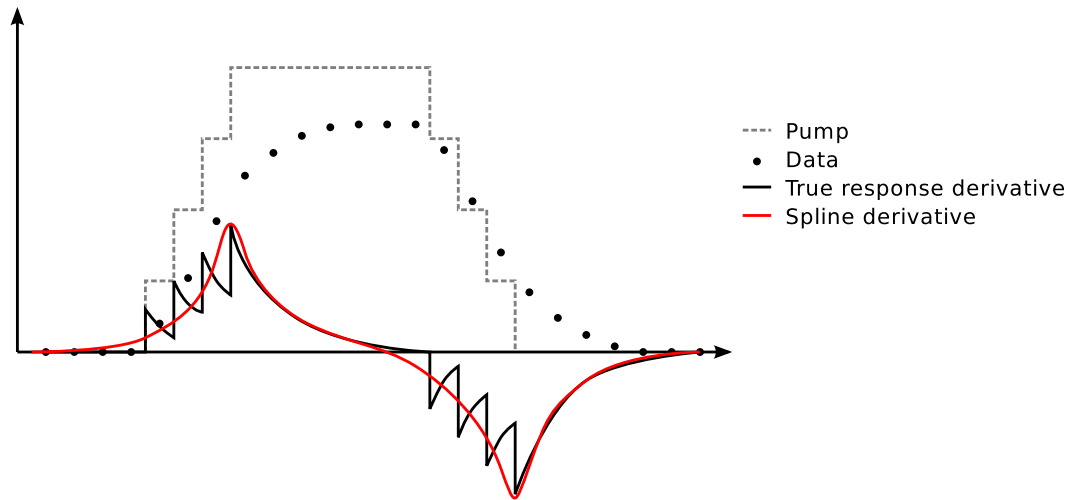


Figure 5.19: Erroneous spline derivatives for trapezoidal waves. The massive discrepancy between the derivatives resulted in equations that did not depend on the pumps.

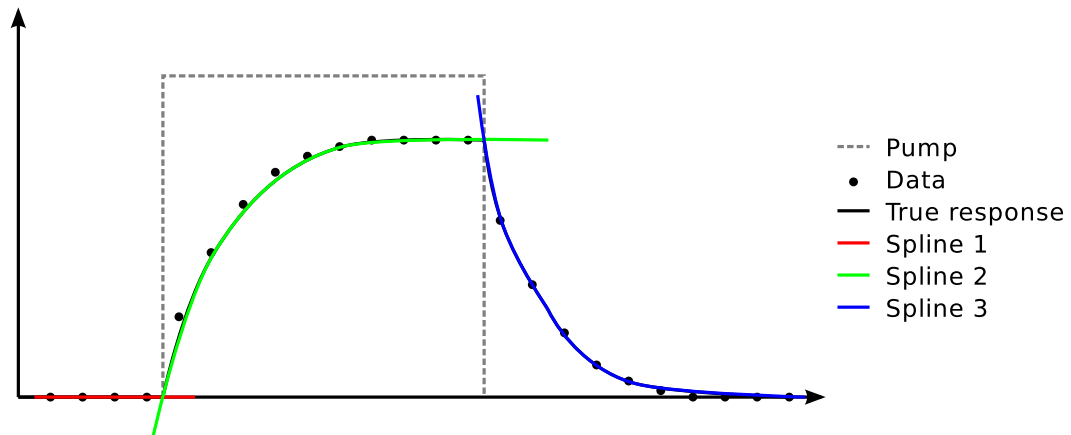


Figure 5.20: Spline fitting with with the data segmented according to the input discontinuities.

of using square waves for the inputs with additional data segmentation was shown to be a functional and accurate protocol. For additional details, refer to Section 5.3.2.

Data weighting

In Section 5.3.5, a major contributor to the error are both steady-state errors as well as one-off spikes. The steady-state error are a systematic issue in the modeling process, while the one-off spikes usually occur near a discontinuity and appear to be measurement noise. Thus, the error calculation was weighted to bias the modeling towards steady-state response and minimize the effect of one-off spikes. Rather than calculating the error as:

$$E = \sum_i |\dot{x}_{i,\text{prediction}} - \dot{x}_{i,\text{data}}| \quad (5.55)$$

the error was calculated as:

$$E = \sum_i w_i |\dot{x}_{i,\text{prediction}} - \dot{x}_{i,\text{data}}| \quad (5.56)$$

where w_i is the weighting term, defined as follows:

$$w_i = \frac{\max_{j \in \text{pump section}} |\dot{x}_{j,\text{data}}| - |\dot{x}_{i,\text{data}}|}{\max_{j \in \text{pump section}} |\dot{x}_{j,\text{data}}|} \quad (5.57)$$

The weighting term effectively scales the error term by the negative magnitude of the signal. When the derivative is at its maximum, which occurs in the transient regime, the error term will be zero regardless of the model prediction. Likewise, when the derivative is zero, or in the steady state regime, the error will take on its full value. The maximum term is calculated for each section where the pumps are constant and is able to deal with the pump discontinuities accordingly.

Model inference via symbolic regression

By following the post-processing steps using a smoothing spline defined in this section, the synchronized time-series data with derivatives is obtained (Fig. 5.21). The data file

is then processed a symbolic regression algorithm – in these experiments, the publicly available Eureqa API [177] was used. For additional details regarding symbolic regression, refer to Section 1.3.2.

t	u1	u2	u3	x1	x2	x3	x4	dx1	dx2	dx3	dx4
0.000	91	0	0	0.00	0.14	0.09	0.09	30.88	16.56	7.96	1.91
0.205	91	32	42	6.33	3.53	1.72	0.48	30.89	16.44	7.94	1.91
0.410	91	32	42	12.62	6.86	3.35	0.87	30.23	16.08	7.90	1.91
0.615	91	32	42	18.64	10.11	4.96	1.26	28.29	15.55	7.83	1.90
0.820	91	32	42	24.13	13.23	6.56	1.65	25.06	14.88	7.74	1.90
1.025	91	32	42	28.86	16.20	8.14	2.04	21.18	14.16	7.61	1.89
1.230	91	32	42	32.87	19.01	9.68	2.43	18.09	13.29	7.47	1.89
1.435	91	32	42	36.33	21.65	11.20	2.82	15.80	12.45	7.31	1.88
1.640	91	32	42	39.41	24.12	12.68	3.20	14.25	11.64	7.14	1.87
1.845	91	32	42	42.18	26.43	14.13	3.59	12.74	10.89	6.95	1.86
2.050	91	32	42	44.63	28.59	15.53	3.97	11.24	10.20	6.76	1.85
2.255	91	32	42	46.79	30.62	16.90	4.35	9.74	9.58	6.57	1.84
2.460	91	32	42	48.65	32.53	18.23	4.72	8.43	9.02	6.37	1.83
2.665	91	32	42	50.26	34.32	19.52	5.10	7.40	8.51	6.17	1.82
2.870	91	32	42	51.70	36.02	20.76	5.47	6.64	8.02	5.97	1.81
3.075	91	32	42	53.00	37.62	21.96	5.84	6.05	7.55	5.75	1.79
3.280	91	32	42	54.18	39.12	23.12	6.21	5.49	7.10	5.54	1.78
3.485	91	32	42	55.25	40.53	24.24	6.58	4.96	6.65	5.34	1.77
3.690	91	32	42	56.22	41.85	25.31	6.94	4.46	6.20	5.16	1.76

Figure 5.21: An example of a data file. This synchronized time-series data with derivatives is obtained by processing the results file in Fig. 5.13 with a smoothing spline.

5.3.4 Results on a local, synthetic system

The first set of experiments were conducted on a synthetic device in an attempt to validate the automated inference algorithms. Compared with the blind device, the ground truth is known for the synthetic device which facilitated debugging and analysis. This series of experiments provided a critical baseline since it confirmed that the inference algorithm is capable of representing the ground truth.

The system was a set of linear, coupled, first-order differential system with additive Gaussian noise defined as:

$$\frac{dx_1}{dt} = 0.006u_1 + 0.01u_2 - 0.01x_1 - 0.02x_2 - 0.01x_3 \quad (5.58)$$

$$\frac{dx_2}{dt} = 0.02x_1 - 0.01x_2 \quad (5.59)$$

$$\frac{dx_3}{dt} = 0.01x_1 - 0.005x_3 - 0.005x_4 \quad (5.60)$$

$$\frac{dx_4}{dt} = 0.005x_3 - 0.0001x_4 \quad (5.61)$$

and the measured data, \hat{x}_i , is defined as $\hat{x}_i = x_i + \mathcal{N}(0, \sigma_N^2)$, where σ_N^2 is the variance of the noise model.

The experiment parameters for the noisy, synthetic system experiment series were:

- Number of experiments in the validation set: 5
- Number of experiments in the validation set: 3 random + 16 designed
- Model inference computation effort: 10^{11} evaluations per location (8 core hours on a 2.8GHz Intel processor)
- Experiment design computation effort: 1000 generations per location (1 core hours on a 2.2GHz Intel processor)
- Building blocks: $\{+, -, \times, \div, e^x, \log(x), x^x\}$

All error figures reported are the root mean squared error normalized by the standard deviation of the validation set:

$$E = \frac{\sqrt{\sum_{n=0}^N (\hat{x}_n - \dot{x}_n)^2}}{\sigma_{\text{validation}}} \quad (5.62)$$

Data collection

The 16 repeatability experiments, 5 experiments in the validation set and 19 experiments in the training set are shown in Fig. 5.22, Fig. 5.24 and Fig. 5.23, respectively.

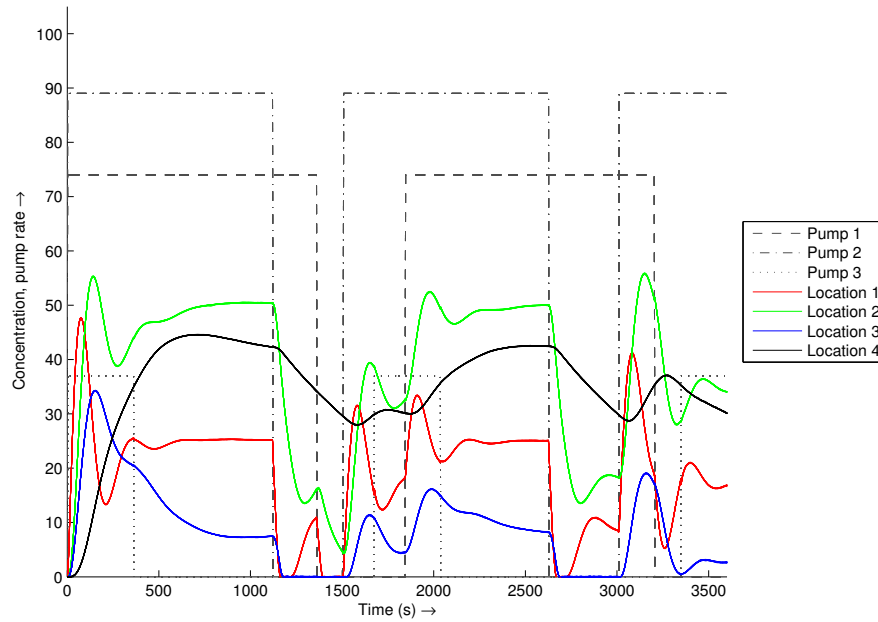


Figure 5.22: The 16 repeatability experiments for the noisy synthetic system overlaid on a single plot.

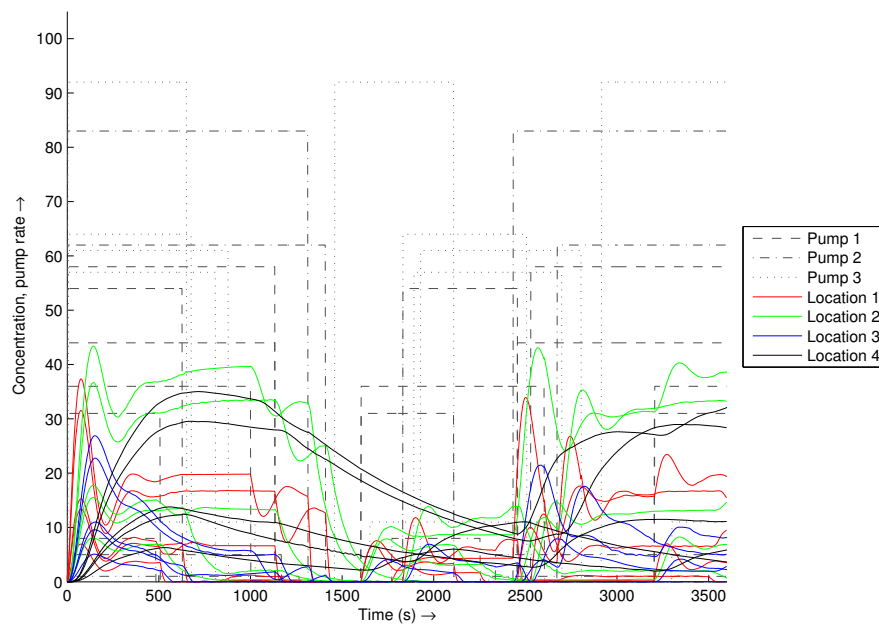


Figure 5.23: The 5 experiments in the validation set for the noisy synthetic system overlaid on a single plot.

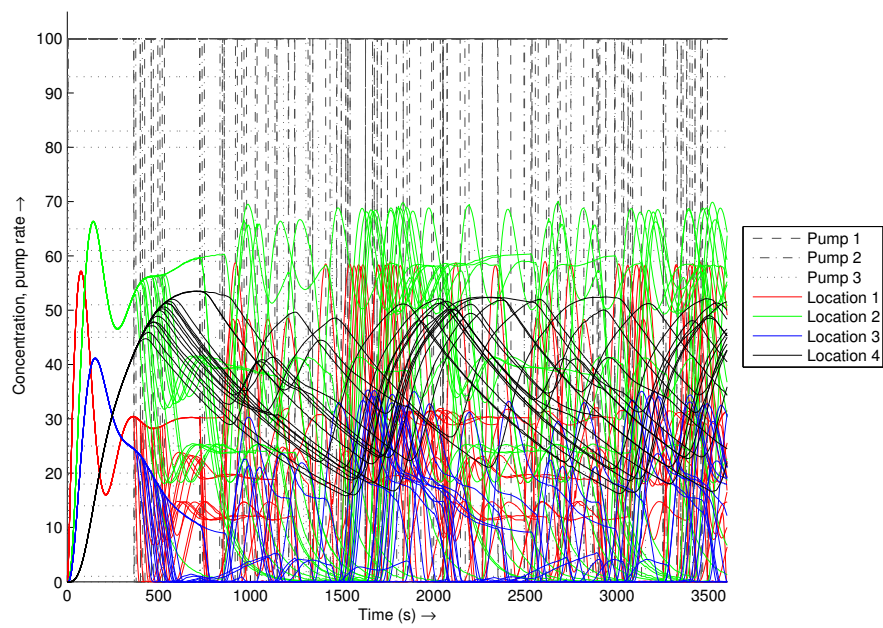


Figure 5.24: The 19 noisy synthetic experiments in the training set overlaid on a single plot.

Analysis

To understand the active learning model inference process, it is important to observe how the candidate models perform as data is collected. Figure 5.25 shows the normalized error of the best model measured on the training data – the plot compares the performance on the training set, which measures how well the equations model the training data, as well its performance on the validation set, which measures of how well the equations generalize to data it has not seen. In comparison, Figure 5.26 shows the normalized error of the best model measured on the validation data – the plot similarly compares the performance on the training and validation set.

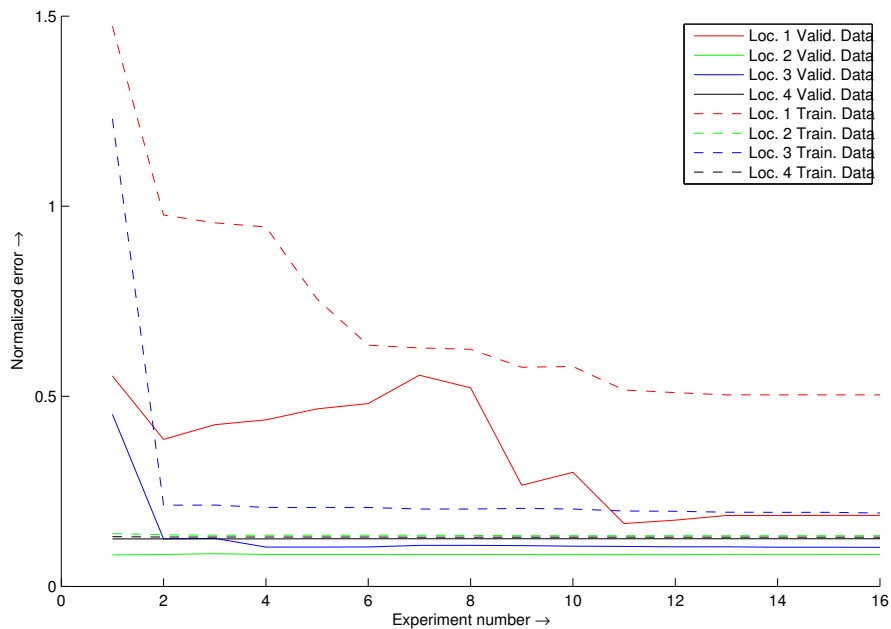


Figure 5.25: The normalized error of the best model measured on the training data for each new noisy synthetic experiments.

Note that the inference algorithm quickly reaches an asymptote for locations 2, 3 and 4. This performance is approximately equivalent to the noise margin in the data and

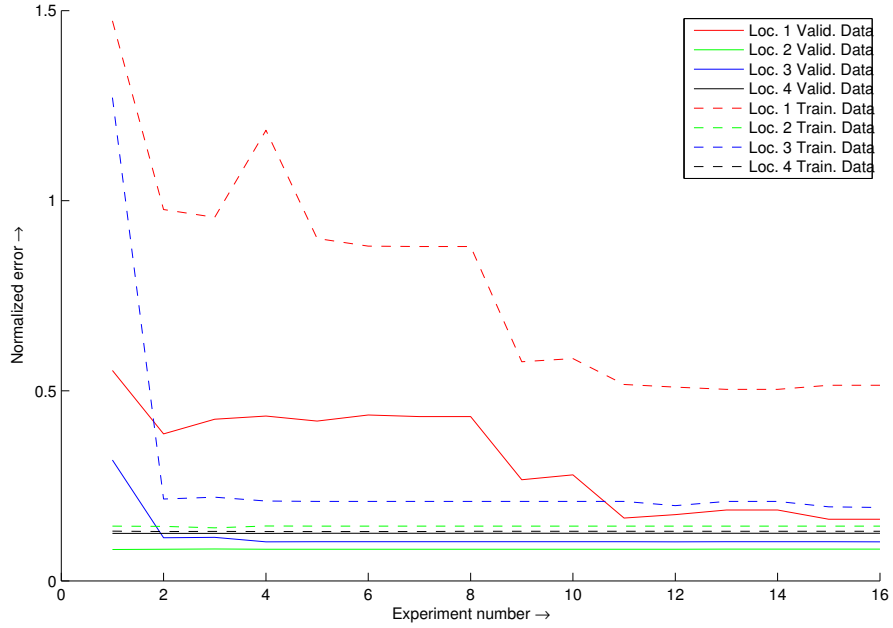


Figure 5.26: The normalized error of the best model measured on the validation data for each new noisy synthetic experiments.

it is impossible to infer higher quality models. Thus, locations 2, 3 and 4 are almost trivial for the inference algorithm to find, even in the presence of noise.

In contrast, the performance on location 1 provides significant insight on the active learning process. The performance from experiments 4 to 8 in Fig. 5.25 shows that model inference algorithm believes it is getting increasing superior models with a marked decrease in training error. However, the validation error actually increases, indicating that these models are poor generalizations and thus they are actually inferior. By comparing Fig. 5.25 and Fig. 5.26 in this region, there is a discrepancy between the best model on the training data and the best model on the validation data.

This trend continues until it reaches an inflection point where the training data contains a sufficient amount of information to describe the dynamics of the system. Both the training and validation error drop drastically at this inflection point, which occurs from

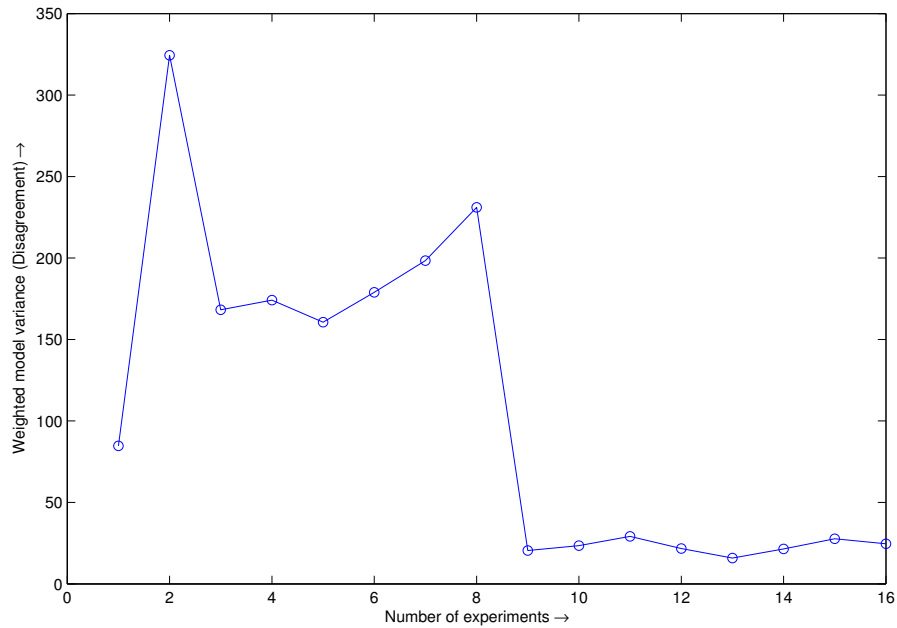


Figure 5.27: The weighted model variance each new experiment in the noisy synthetic experiments.

experiment 9 to 11. At this point the best models for the training and validation data are the same model. Although the models reach a relatively high error asymptote, they predict an accurate approximation of the ground truth and only parameter optimization is left.

Figure 5.27 shows the weighted model variance with respect to each experiment. The weighted model variance is a measure of disagreement between the candidate models. At experiment 9, there is a sharp decrease in the model variance suggesting that the inferred models have converged on a set of candidate models.

The analysis compares two different models: the *best validation models* are the equations that produced the lowest validation error, regardless of complexity or interpretability, and the *linear regression models* are determined via least squared error regression and is used as a baseline. The performance of these two models are summarized in Ta-

Table 5.5: Model inference performance on the validation set for the noisy synthetic experiments.

Location	Normalized error		
	[% of $\sigma_{\text{validation}}$]		
	System Noise	Best Validation	Linear Regression
1	4.40	16.20	40.21
2	6.75	8.36	8.28
3	9.36	10.28	39.90
4	11.52	12.51	12.47

ble 5.5, along with the system noise measured from the repeatability experiment, which provides an experimental limit on the performance.

The best validation model performs comparably to the system noise for locations 2, 3 and 4. Although location 1 has a relatively high error, it correctly predicts the ground truth and only additional parameter optimization with more data is required. In comparison, linear regression performed significantly worse on both location 1 and 4. This is an interesting result given that the ground truth is a linear model and linear regression is capable of modelling such systems.

The best validation models are:

$$\frac{dx_1}{dt} = 0.5888u_1 + 0.01153u_2 - 0.009414x_1 - 0.01976x_2 - 0.01015x_3 \quad (5.63)$$

$$\frac{dx_2}{dt} = 0.01995x_1 - 0.009825x_2 - 0.0001539x_4 \quad (5.64)$$

$$\frac{dx_3}{dt} = 0.009972x_1 - 0.004786x_3 - 0.005064x_4 \quad (5.65)$$

$$\frac{ds_4}{dt} = 0.005030x_3 - 0.0009981x_4 \quad (5.66)$$

It is clear that these models are close approximations to the ground truth equations (Eq. 5.58-5.61) and the difference in the equations are due to the presence of noise in the data set. Thus, this experiment series confirmed that the inference algorithm is capable of successfully inferring the correct model from a numerical and symbolic perspective. Furthermore, the performance of the active learning algorithm with the surprisal optimization criterion provides a baseline of what to expect when the inference algorithm converges to a consistent model.

5.3.5 Remote operated experiments on a physical device

This section presents a set of experiments were conducted on the physical microfluidic device described in Section 5.3.1.

Experimental setup

The experiment parameters for the microfluidic device experiment series were:

- Number of experiments in the validation set: 8 random experiments
- Number of experiments in the training set: 16; 3 random experiments + 13 designed experiments
- Model inference computation effort: 10^{11} evaluations per location (8 core hours on a 2.8GHz Intel processor)
- Experiment design computation effort: 1000 generations per location (1 core hours on a 2.2GHz Intel processor)
- Building blocks: $\{+, -, \times, \div, x^y\}$

Data collection

The 24 repeatability experiments, 8 experiments in the validation set and 16 experiments in the training set are shown in Fig. 5.28, Fig. 5.29 and Fig. 5.30, respectively.

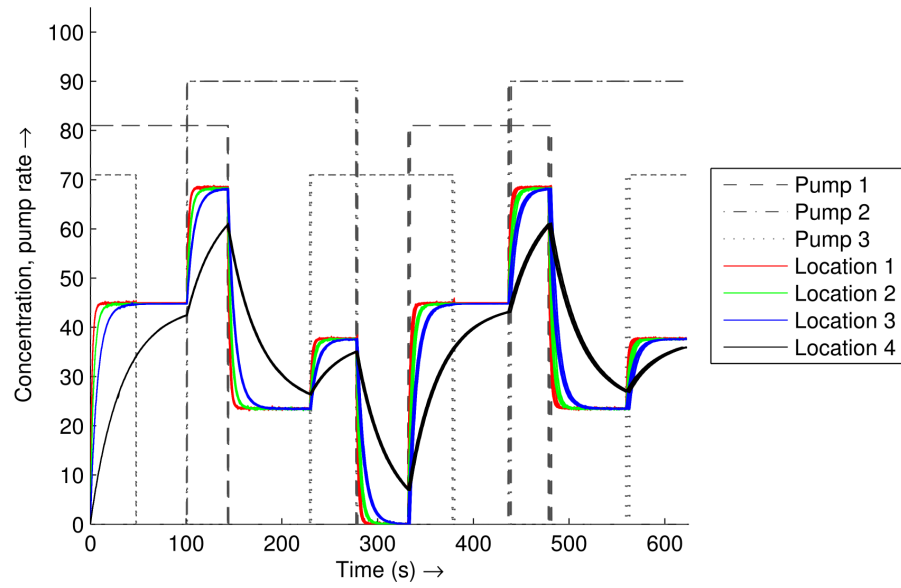


Figure 5.28: The 24 repeatability experiments for the microfluidic device overlaid on a single plot.

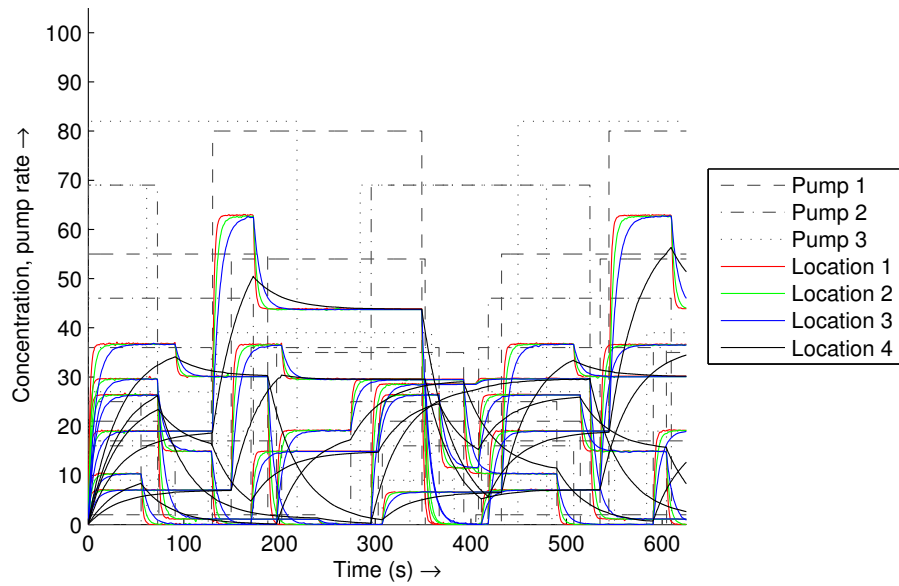


Figure 5.29: The 8 experiments in the training set for the microfluidic device overlaid on a single plot.

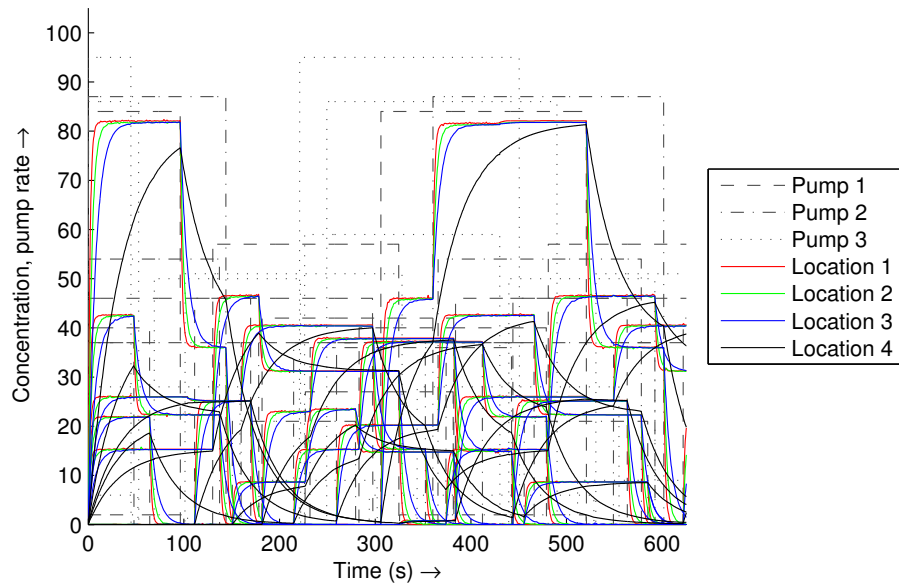


Figure 5.30: The 16 experiments in the validation set for the microfluidic device overlaid on a single plot.

Analysis

Beginning with three initial random experiments, the active learning algorithm iterated between the two processes of model inference and experiment design. The inferred models were used to create predictions, which in turn were compared against a validation data set of eight randomly generated experiments. The validation set was not used for training or model inference.

The performance of the learning algorithm is best quantified by the training and validation errors. Normalizing the error by the standard deviation, Table 5.6 summarizes the performance of the algorithms, as well as the measurement noise determined from the repeatability experiments.

	Normalized Error [% of Standard Deviation]			
	x_1	x_2	x_3	x_4
Training error	5.6	5.2	4.3	5.4
Validation error	7.6	6.5	5.1	3.8
System noise	8.3	5.6	3.8	3.6

Table 5.6: The error on the training and validation data sets, as well as the system noise for repeated experiments.

In Table 5.6, the validation error is within the bounds of the system noise. The system noise defines a lower bound on the repeatability of the experiments, and it is impossible to infer a model that achieves a lower error than the inherent measurement noise in the system. Furthermore, the training and validation error are both consistent and have reached the bounds of the experimental setup, which suggests that the algorithm has found a predictive model of the system.

The inferred model of the microfluidic device consists of four, first order differential equations:

$$\frac{dx_1}{dt} = 0.3212u_1 + 0.2424u_2 - 0.5864x_1 + 0.01308 \quad (5.67)$$

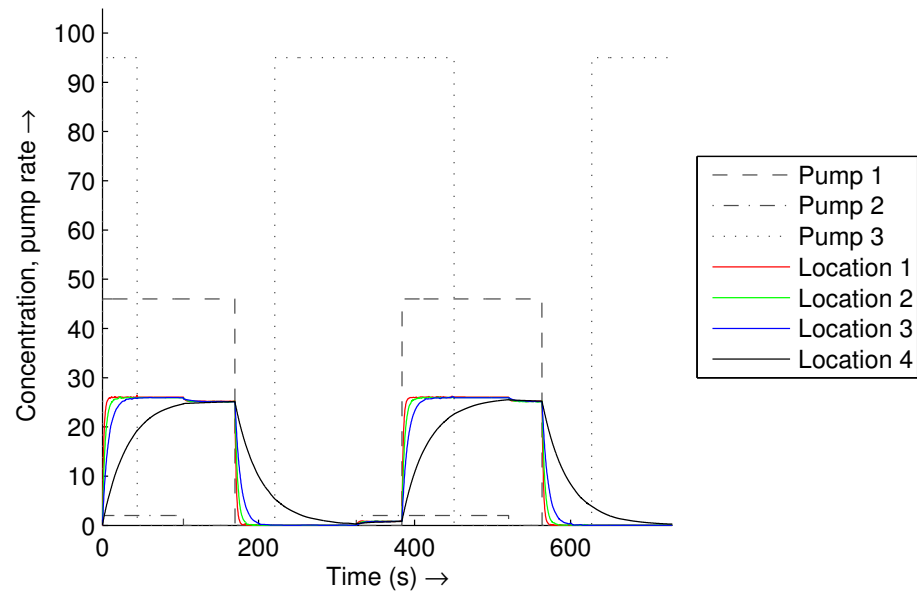
$$\frac{dx_2}{dt} = 0.1568u_1 + 0.1182u_2 - 0.2870x_2 \quad (5.68)$$

$$\frac{dx_3}{dt} = 0.07431u_1 + 0.05602u_2 - 0.1361x_3 + 0.003429 \quad (5.69)$$

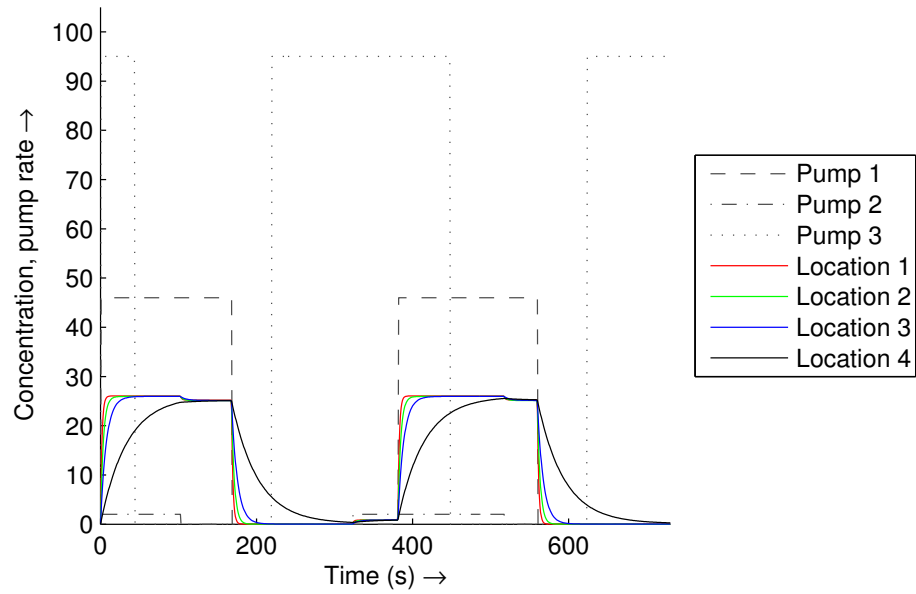
$$\frac{dx_4}{dt} = 0.01627u_1 + 0.01224u_2 - 0.02988x_4 + 0.003436 \quad (5.70)$$

where x_i is the i -th location state and u_j is the j -th input pump rate. This type of model agrees with the intuitive expectations of the square-wave step response observed in Figs. 5.28-5.30.

Using these equations, simulations of the validation data can be generated. By using the input pump parameters of the validation, the differential equations were numerically integrated and compared against the corresponding recorded data. The algorithm was not trained on the validation data and its performance on the validation set provides a measure of its ability to generalize and describe the underlying dynamics. The measured validation data and the corresponding the simulation the model (Eq. 5.67-5.70) are shown in Fig. 5.31-5.38 for each validation experiment, respectively. The simulations are visually indistinguishable to the measured experiments, with the exception of measurement noise.

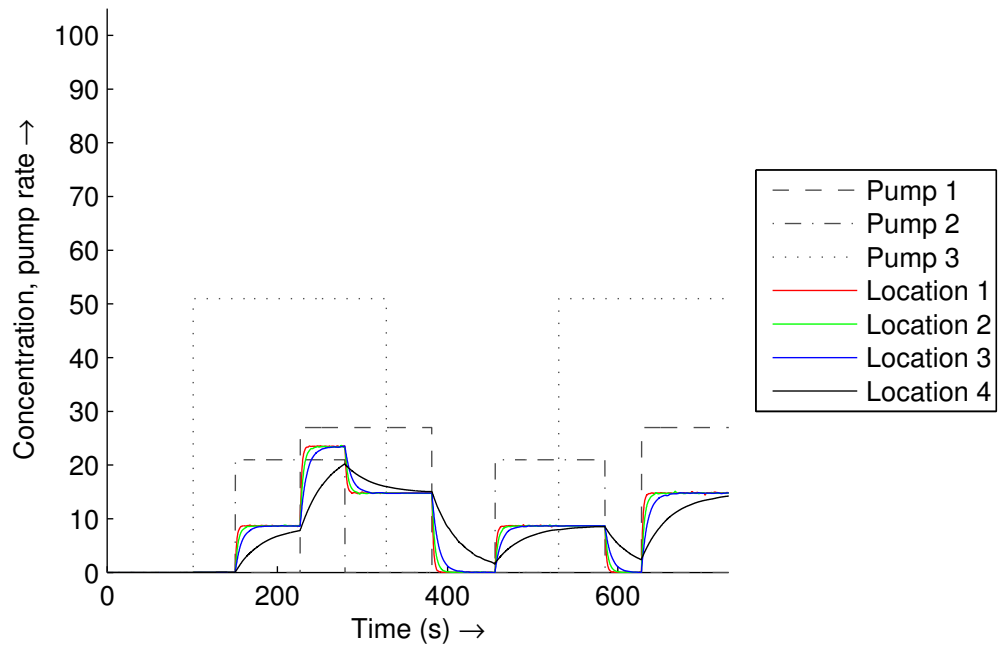


(a) Measured validation data

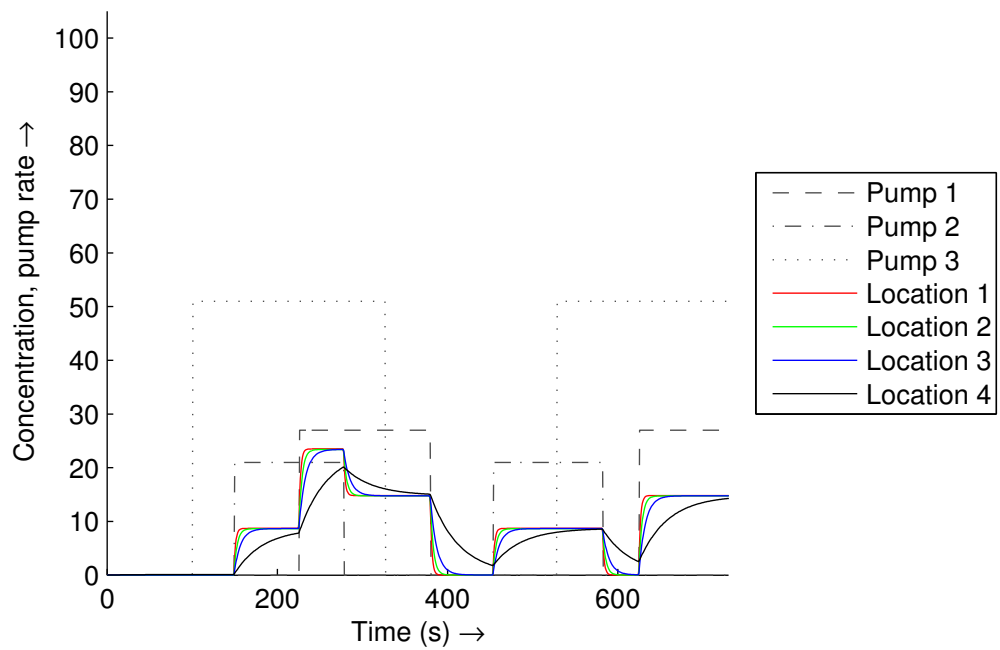


(b) Simulated model given the same inputs

Figure 5.31: Validation experiment 0.

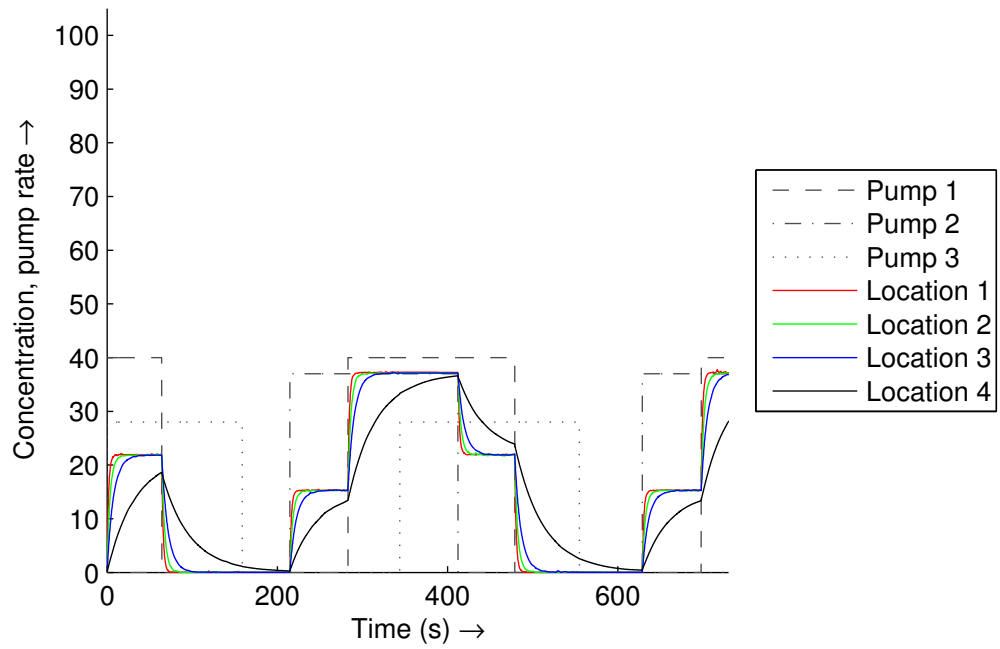


(a) Measured validation data

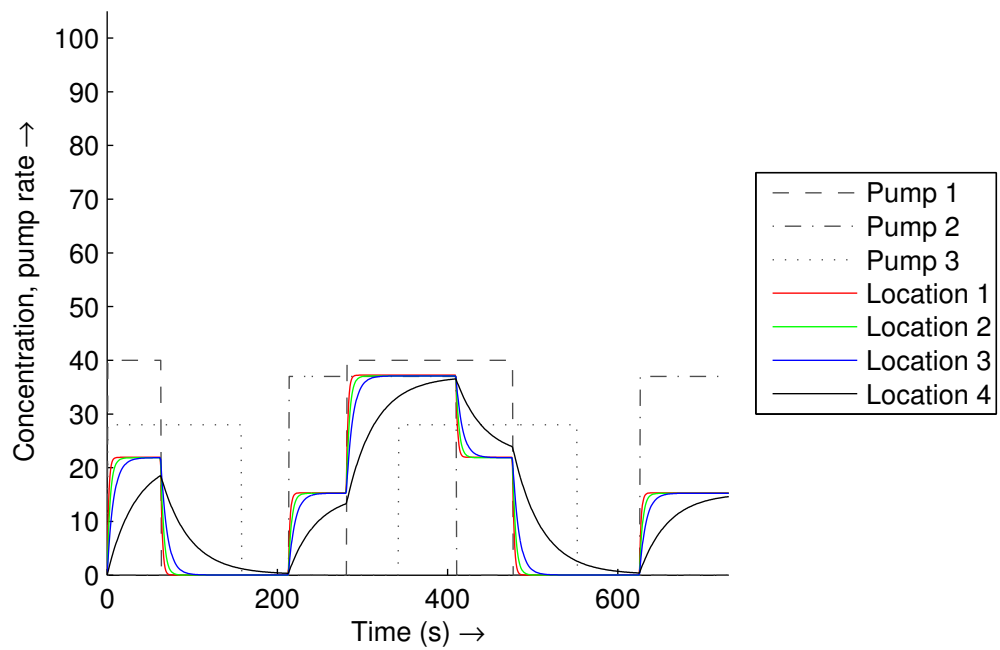


(b) Simulated model given the same inputs

Figure 5.32: Validation experiment 1.

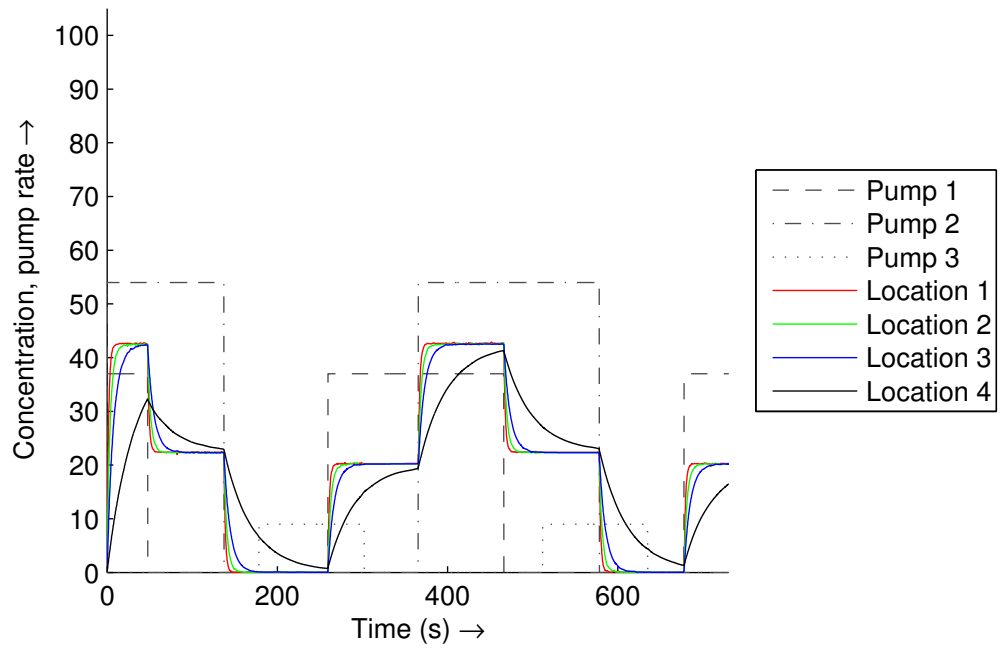


(a) Measured validation data

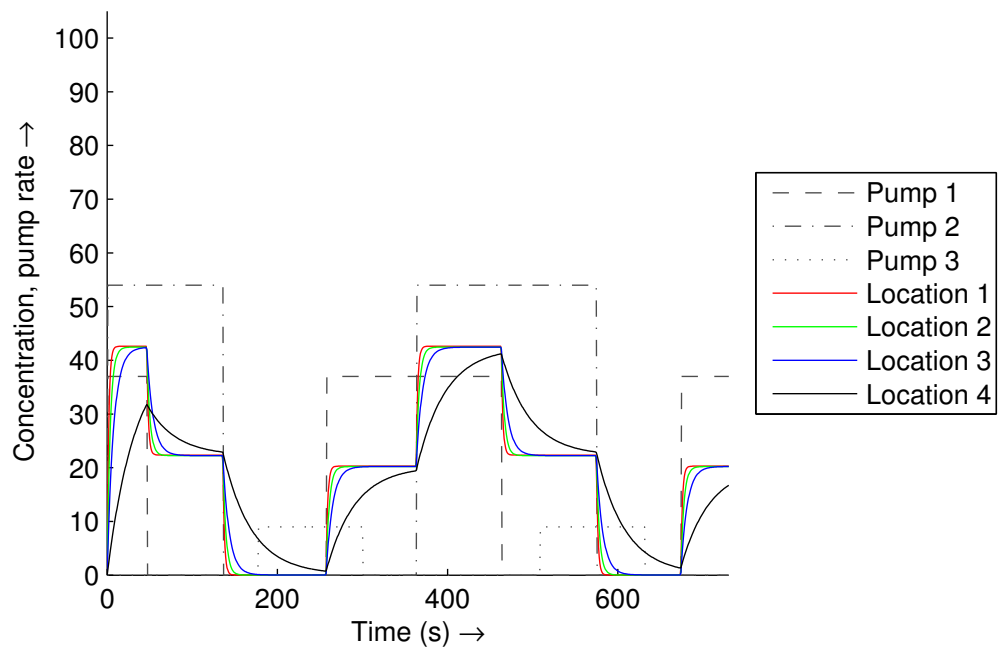


(b) Simulated model given the same inputs

Figure 5.33: Validation experiment 2.

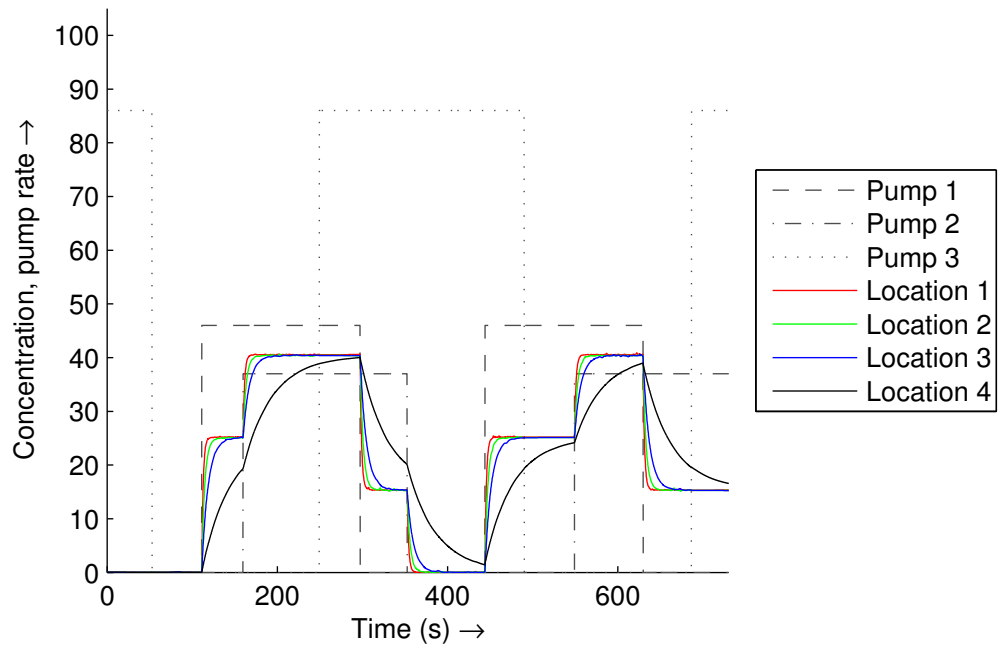


(a) Measured validation data

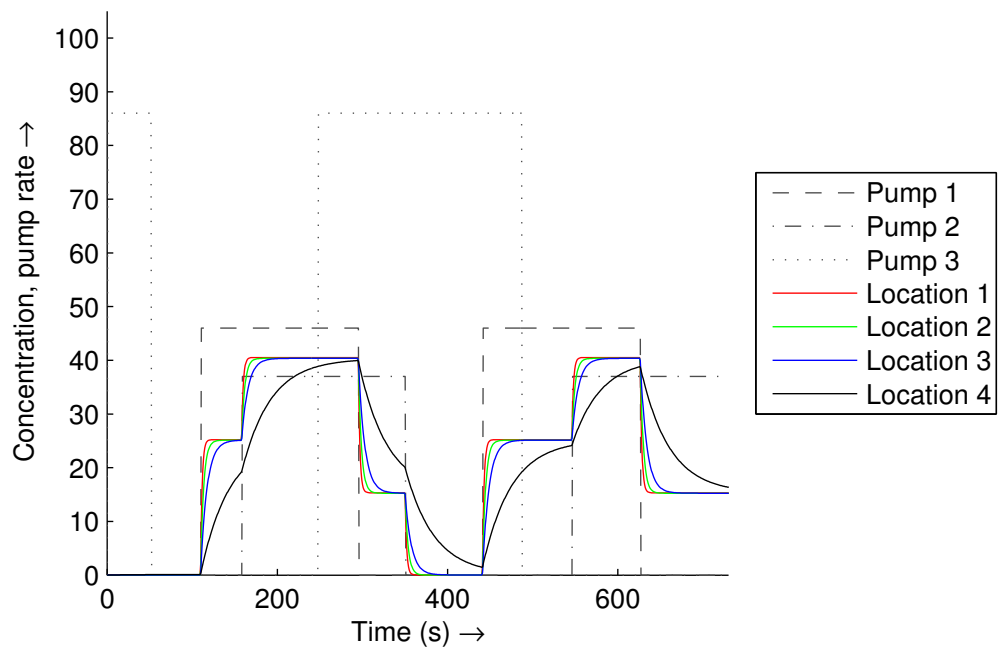


(b) Simulated model given the same inputs

Figure 5.34: Validation experiment 3.

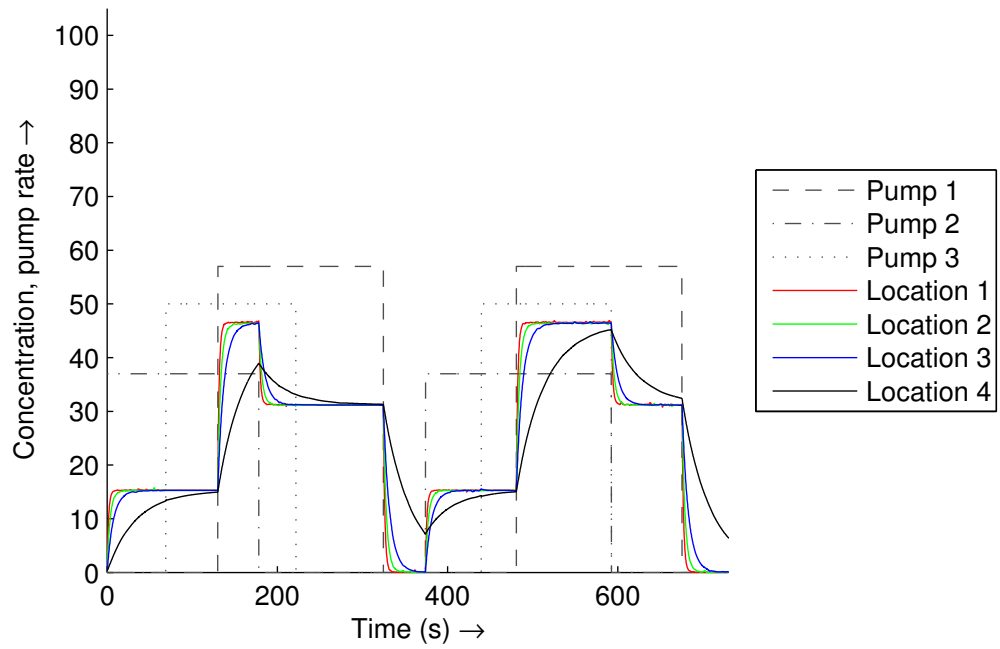


(a) Measured validation data

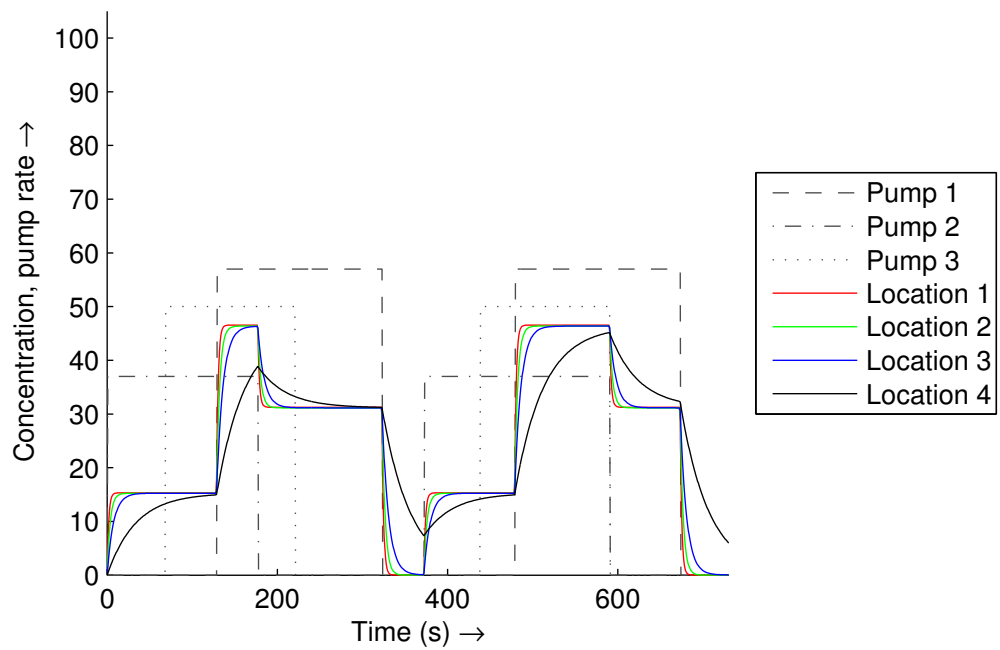


(b) Simulated model given the same inputs

Figure 5.35: Validation experiment 4.

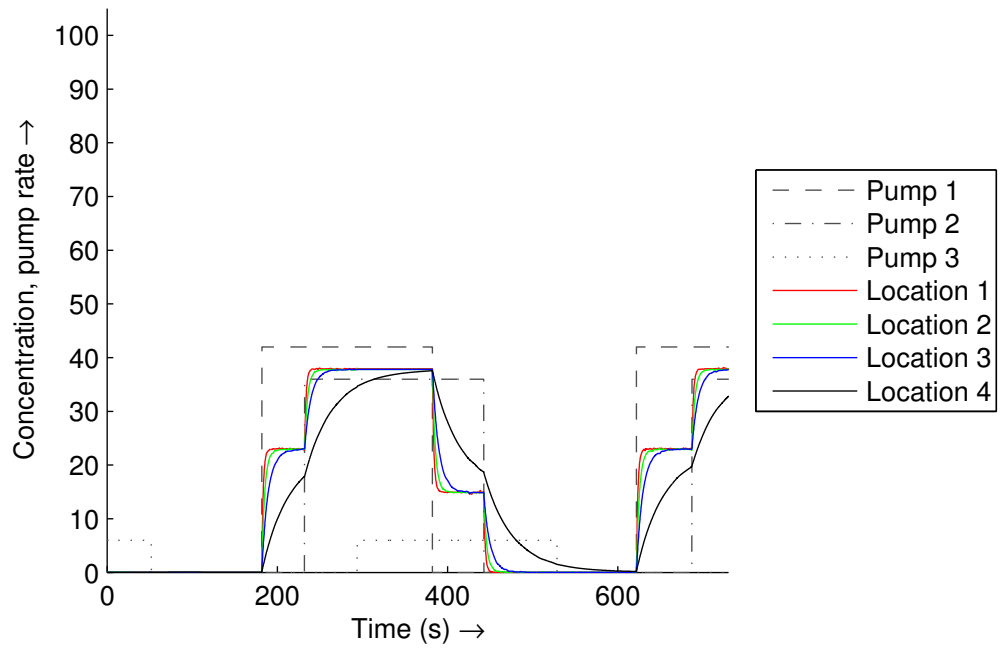


(a) Measured validation data

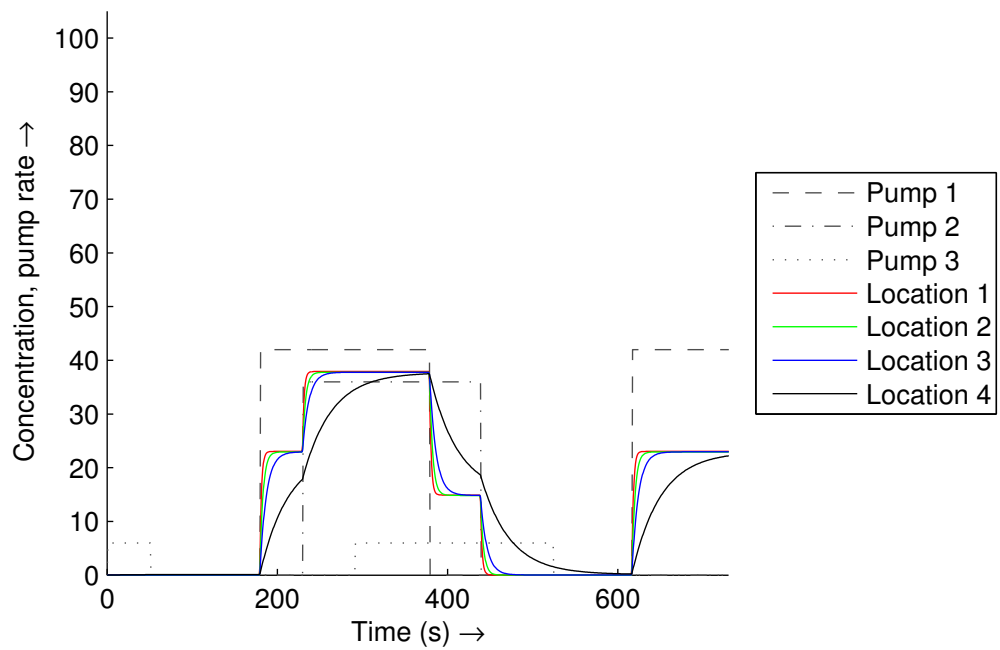


(b) Simulated model given the same inputs

Figure 5.36: Validation experiment 5.

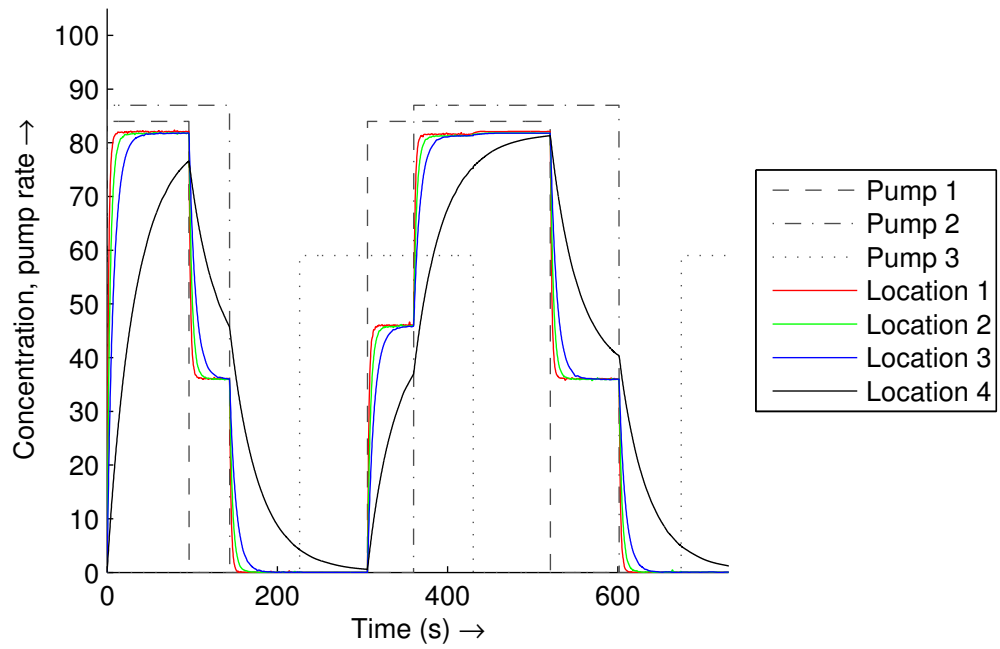


(a) Measured validation data

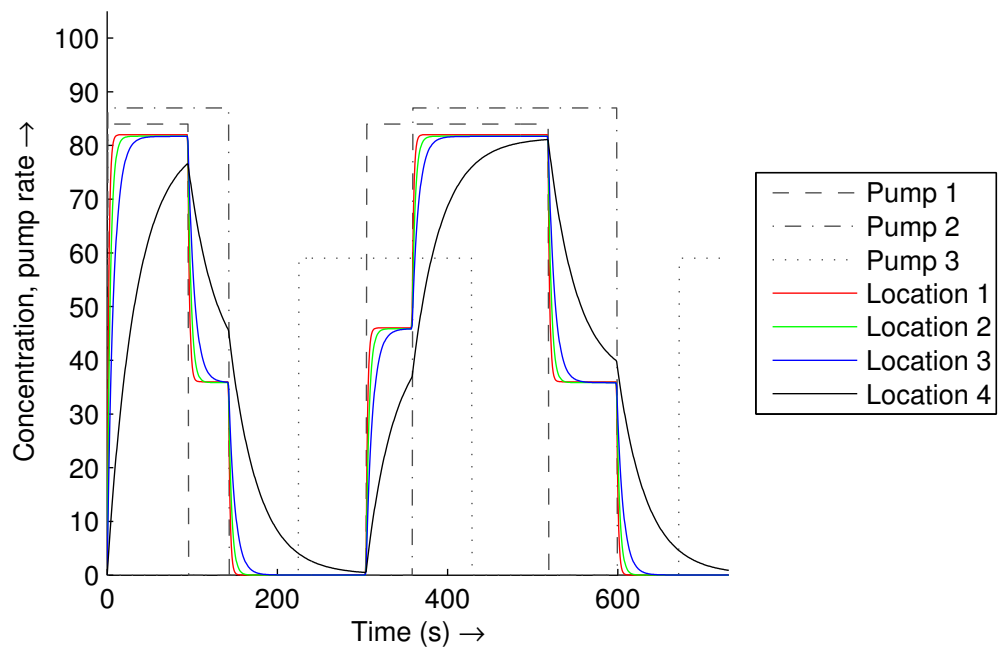


(b) Simulated model given the same inputs

Figure 5.37: Validation experiment 6.



(a) Measured validation data



(b) Simulated model given the same inputs

Figure 5.38: Validation experiment 7.

5.3.6 Conclusions and future work

An active machine learning architecture for the remote, automated inference of dynamical systems from systematic probing was demonstrated. The architecture discovered coupled, first-order system dynamics from a black-box device by automating the components of physical experimentation, scientific modeling and experimental design. The device was designed as a physical analogy of a multi-compartmental pharmacokinetic model. Through the remote operation of a pump-microscope assembly, this work illustrates how a set of distributed machines can accelerate the modeling process and contribute scientific knowledge while scaling beyond geographic constraints.

As this is the first stage in a long-term collaborative project, there are many routes for future work. From a machine learning perspective, a ripe area for future research is the investigation of more complex experiment design. The square-wave approach was selected due to its ease of implementation and analysis. However, different classes of inputs elicit drastically different information from dynamical systems and, thus, it is important to consider alternatives to the experiment design.

From an experimental perspective, the success of the microfluidic analogy paves the way for investigating more complex, physical phenomena. Rather than studying the interactions of dyes within a microfluidic chamber, biological or chemical systems can be analyzed in a microfluidic device using the same infrastructure. This research is the first step towards automated scientific discovery where new scientific knowledge and theories are produced solely from an automated system without human intervention.

CHAPTER 6

LIMITATIONS AND FUTURE WORK

This chapter discusses the limitations and constraints of the algorithms and approaches presented in this dissertation. Extending the capabilities of these methods to address the listed limitations is a prime area for future work and may lead to critical advancements to the field of automated science.

6.1 Evolutionary computation (Section 1.3.1)

Although evolutionary computation is not a novel contribution of this dissertation, the vast majority of algorithms presented in this work use some variation of evolutionary computation as a core component of its processes. As a result, it is important to discuss the limitations of evolutionary computation as these limitations also extend to the other algorithms in this work through transitivity.

1. **Heuristic approach** - Perhaps the single most critical issue in the use of evolutionary computation for automated science is that evolutionary computation is fundamentally a heuristic approach. Although they have been shown to address difficult optimization and design problems, the heuristic approach leads to several unfavorable properties:

- Lack of optimality - heuristic approaches cannot guarantee that it will be able to find the global or local optimal solution
- Lack of completeness - heuristic approaches cannot guarantee that it will find a solution if one exists, and if one does not exist, cannot report that no solution is possible

- Lack of performance guarantees with respect to computational effort - heuristic approaches cannot guarantee the performance of its candidate solution as a function of computational effort

Thus, many of the experimental parameters in this work were selected through user experience and domain expertise. A critical area for future work is to prove some bounds on the performance of evolutionary computation such that the candidate solutions can be stated within a confidence interval.

2. **Choice of representation** - The performance of an evolutionary algorithm depends highly on the genotypic representation of candidate solutions. Different representations provide different tradeoffs for each problem [42, 128, 171, 138], but these tradeoffs are difficult to predict in advance in any rigorous manner. The choice of representation in this work was selected through user experience and ease of implementation.
3. **Choice of evolutionary operators** - A fundamental concept in evolutionary computation is that the evolutionary operators, generally recombination and crossover, produces new individuals that are small variations of their parents. While it is easy to ensure that the small variations exist in the genotypic or representation space, it is significantly more challenging to provide the same guarantees in the phenotypic or fitness space. From previous experience, this limitation did not prohibit the evolutionary algorithms from finding suitable solutions and thus, was not directly addressed in this work.
4. **Choice of building blocks** - Closely related to the issues of ‘the choice of representation’, the choice of building blocks has a critical affect on the performance of the evolutionary algorithm. Due to the lack of completeness, it is difficult to know if the evolutionary algorithm has been provided with all the necessary building blocks to be able to even represent the solution. In some specific cases, it has

been shown that evolutionary algorithms are capable of finding accurate approximations; for example, a symbolic regression algorithm was able to find Taylor expansions of $\sin(x)$ when it lacked any trigonometric building blocks [173]. Many of the experiments in this work were selected to avoid this issue and is left for future work.

5. **Choice of fitness metric** - The choice of a fitness metric, which evaluates how well the solution fits the observed data, directly affects the computational effort of the search. For example, using a logarithmic error instead of a squared error can result in drastically different convergence times. The fitness metric in this work was generally selected through user experience and domain expertise, and methods to automate this process is left for future work.

6.2 Trainer selection strategies (Section 2.1)

This work presented a case study on different trainer selection strategies for rank prediction coevolution. This work used variance as its basis for a fitness metric, which assumes that the trainer's ranking can be appropriately modeled as either a discrete or continuous random variable. A more natural representation, such as one that uses ranking distributions [1], might lead to more suitable measures for trainer selection.

6.3 Optimal experiment design (Section 2.2)

This work presented a method for non-parametric optimal experiment design for co-evolutionary systems based on Shannon information. There are several presumptive constraints and avenues for future work:

1. **Well-defined experiment design** - The design of experiments in this work requires that the space of all possible experiments is well-defined. In this work, the experiments is a vector of real numbers that correspond to experimental inputs, but the work can be readily extended to design experiments with a more complex representation such as ones with tree structures. Nonetheless, the experiment design algorithm is constricted to this well-defined space and cannot design novel types of experiment. Often key scientific breakthroughs required the design of novel instruments that revolutionized experimental protocols and provided new perspectives to analyze data. Designing methods to automate this process is an important area of research for the future of automated science.
2. **Designing a series of experiments** - The current approach to the design of experiments assumes a serial processing through the iterations of experiment design, physical experimentation and model inference. As a result, this work investigates the optimal design of a single experiment that best disambiguates the competing models. However, there is the possibility for experimental systems to execute many experiments in parallel, which allows for the potential for faster scientific discovery. Thus, experimental design should be extended to these parallel systems and allow for the design of a collection of experiments that maximize the mutual information content.
3. **Experimental work using Shannon entropy** - Although this work showed that the surprisal metric approximates entropy within a bounded error, it would be a meaningful to show how the two metrics for experiment design compared experimentally across a series of baselines.

6.4 Kinematic pose inference from RGBD images (Section 2.3)

This work presented a method to infer poses from RGBD images based on kinematic skeletons. Since the algorithm uses evolutionary computation, the limitations described in Section 6.1 also apply this algorithm. More specifically, there are two limiting constraints for kinematic pose inference:

1. **Space-filling representation** - The algorithm aims to generate a pose of the skeleton that generates the observed depth image (Section 2.3.2), which implicitly assumes that the kinematic model is able to fill the space of the point cloud. An acyclic graph structure of rigid links, which consists of hemisphere and cylinder components, was chosen due its ease of parameterization and ease of fitness computation. However, this structure is only a rough approximation of to physical bodies. Details including links with non-uniform thickness, space filling joints and complexities introduced by clothing are not captured by the hemisphere and cylinder representation. Representations that better fill the space in depth images could result in superior model inference.
2. **The skeleton structure is known in advance** - The algorithm assumes that the given skeleton is suitable for the depth image and relies on the end user to know which skeletons are appropriate in advance. Rather than assuming that the skeleton is known, the acyclic graph structure can be included as part of the search and a genetic programming approach can be used to find the skeleton and its corresponding pose simultaneously.

6.5 Clustered symbolic regression (Section 3.2.2)

This work presented an unsupervised method to cluster input-output data into symbolic functions. Since the algorithm uses evolutionary computation, the limitations described in Section 6.1 also apply to this algorithm. More specifically, there are three limiting constraints for clustered symbolic regression:

1. **Additive noise model** - The algorithm assumes an additive Gaussian noise model. Although a Gaussian model is appropriate as a model of measurement noise, it is not obvious how the algorithm can be readily adapted to other noise models. Furthermore, the algorithm builds on the noise model in a heuristic fashion and does not develop a strong theoretical approach that integrates the noise model into the algorithm. Thus, a framework that is derived from first principles could help provide a more stable basis for clustered symbolic regression.
2. **Rich data sets** - The experimental work on clustered symbolic regression used rich data sets with thousands of points in each subfunction. It was assumed that the data sets would not be sparse, and that there would be enough data to statistically test the validity of the hypothesized clusters. Future work includes investigating how the algorithm performs as there is less data, as well as if there is a non-uniform distribution of data points in each cluster.
3. **Locally optimal clustering** - The algorithm depends on the expectation-maximization algorithm, which is a locally optimal algorithm, as well as symbolic regression, which is a heuristic approach. The combination of these two approaches makes it difficult to provide any guarantees on performance. In practice, it was shown that the global optimal was frequently found but only after a significant amount of computational effort, which was not known in advance.

and was determined experimentally. Thus, a more general framework to predict when the algorithm has reached a locally optimal solution is important to finding clustered symbolic functions.

6.6 Inferring symbolic binary classifiers (Section 3.3)

This work presented a method to infer into binary classification boundaries as symbolic expressions. Since the algorithm uses evolutionary computation, the limitations described in Section 6.1 also apply this algorithm. More specifically, there are two limiting constraints for modeling transition conditions:

1. **Theoretical framework** - The algorithm uses an absolute objective error to infer symbolic expressions. This error was chosen due to its ease of implementation and to show how a single symbolic regression framework could be use for both regression and classification inference. However, there is a lack of theoretical motivation for this choice of objective error. Other historical approaches, including logistic classification and maximum-margin classifiers, could provide a more robust approach to symbolic classification.
2. **Multi-class classification** - The algorithm was formulated as a collection of binary classification problems, due to its similarity to the symbolic regression problem. However, an algorithm that solves the multi-class classification problem, where mutual exclusivity is guaranteed, is more suitable for this problem and presents a significantly greater challenge from a machine learning perspective.

6.7 Inference of hybrid automata (Section 3.2)

This work presented a method to infer symbolic models of discrete-continuous hybrid automata. The algorithm is built from a series of approaches, including evolutionary computation (Section 6.1), clustered symbolic regression (Section 6.5), and symbolic binary classifiers (Section 6.6), and their respective limitations also apply this algorithm.

In addition, the algorithm requires several important constraints to be met in advance (Section 3.1.2: each behavior is unique, there are no continuous state variables, and the number of discrete modes is known. These constraints define a very specific type of discrete-continuous hybrid system and developing new algorithms that extend the inference capabilities beyond these constraints is critical to automated science.

6.8 Inference of ordinary differential equations of arbitrary order (Section 4.2)

This work presented an unsupervised method to infer ordinary differential equations of arbitrary order from time-series data. Since the algorithm uses evolutionary computation, the limitations described in Section 6.1 also apply this algorithm. More specifically, there are two limiting constraints for this algorithm:

1. **Additive noise model** - The algorithm assumes a measurement noise model – specifically, the state evolution is noiseless and there is only additive noise on the observations. Although this approach is common and relatively useful in practice, it is nonetheless a very simple noise model. In practice, the algorithm was able to still find the correct model with 10% noise, measured with respect to the stan-

dard deviation of the signal, which was approximately the limit of the algorithm's performance.

However, since the underlying process is a ordinary differential equation, there is a larger class of differential equations with stochastic properties that are unsuitable for this algorithm. For example, systems that include Wiener processes or systems that are described by Langevin or Gillespie equations are ill-suited for this approach. Thus, the type of noise, as well as the magnitude of the noise, has a large effect on the performance of the algorithm.

2. **Rich data sets** - The experimental work used rich data sets with a sampling frequency significantly higher than the Nyquist frequency. In fact, high frequency samples was critical for the accurate calculation of higher order derivatives, an important part of the inference algorithm. Understanding how the algorithm performs with relatively low frequency samples is an important avenue for future work.

6.9 Discovering simple representations of dynamical systems (Section 4.3)

This work presented an unsupervised method to discover simple representations of dynamical systems. Since the algorithm uses evolutionary computation, the limitations described in Section 6.1 also apply this algorithm. More specifically, there are three limiting constraints for this algorithm:

1. **Relies on the inference of higher order ordinary differential equations** - As described in Section 4.1, the process to discover hidden variables first depends

on inferring higher order ordinary differential equations from time-series data. Once an appropriate differential equation model is found, then additional symbolic analysis is conducted in this algorithm to find simple representations that are candidates for hidden variables. Thus, by proxy, all the limitations described in Section 6.8 are indirect limitations to this algorithm.

2. **Scalar multiplicatives of hidden variables** - The search for meaningful hidden variables is driven by the principle of simplicity. However, a primary constraint is that the hidden variable itself can be multiplied by any scalar while retaining the same structural form. Thus, the search is unable to determine the exact scalar multiplicative for the hidden variable itself.
3. **Locally optimal solutions** - The search for meaningful hidden variables can be challenging due to the possible existence of locally optimal solutions. Simple candidate solutions occur when there are algebraic cancellations. Since the search error weighs each component by the size of the corresponding subtree, this may result in a search space with many local optima, which drastically increases the difficulty of the search.

6.10 Inferring luminescent chemical reactions (Section 5.2)

This work presented preliminary research on the inferring hidden structure from the luminescent reactions from the luciferase and TCPO chemicals. This work was in its early stages and would be best advanced by further investigation and experimentation on physical systems.

6.11 Inferring multi-compartment pharmacokinetics models (Section 5.3)

This work presented a real-world example of automated telescience and the machine discovery of multi-compartment pharmacokinetics model from physical systems. There are several limiting constraints for this approach for automated science:

1. **Bounded experiment space** - A major subcomponent of this work is optimal experiment design, and thus the limitations summarized in Section 6.3 also apply to this research.

In addition, another constraint for physical systems is that the device must be designed in advance with the bounds of the experiment in consideration. Candidate experiments can only be designed within the space of physically executable experiments, and thus the space must include experiments that elicit the intended behavior. These bounds must be known in advance so that the system can be constructed.

2. **Precise actuation and data acquisition** - Inferring dynamical systems requires significant amounts of time-series data, which is only made possible by physical devices with precise actuation and data acquisition. As described in Section 6.8, rich data sets with sampling frequencies significantly higher than the Nyquist frequency are required and can only be obtained experimentally with the appropriate instruments.
3. **Theoretical framework for experimental design of differential models** - For experimental design of differential models, an approach based on using the weighted average of the simulated signals was used in Section 5.3.2. This approach was based on intuition and experience and it is not clear how to adapt this

approach to other problems. A stronger theoretical framework that provides an explanation for experimental design of differential models would provide greater generalization for applications in other fields.

4. **Model inference** - A major subcomponent of this work is model inference based on evolutionary computation, and thus the limitations summarized in Section 6.1 also apply to this research. In particular, the issue of the choice of building blocks is particularly pertinent for the inference of multi-compartment pharmacokinetics model since complex dynamics may not be representable by the traditional building blocks.
5. **Internal statistical analysis** - An area for future development is greater integration with the statistical tools already in the algorithm for more consistent analysis. For example, adding additional knots to the smoothing splines at input discontinuities allows one to control the discontinuity of a single spline. Furthermore, the cross-validation error for fitting the smoothing spline also provides information regarding the measurement noise and could be used for additional analysis.

CONTRIBUTIONS

Major contributions

Chapter 2: Coevolutionary search methods and applications

- **Presented the first study for trainer selection strategies in rank prediction coevolutionary systems.** Previous work in this field focused on developing heuristics for evolving predictors [172, 176], but did not systematically explore the dynamics of trainer selection strategies.
- **Demonstrated the use of surprisal of the mean for optimal experiment design for nonparametric active machine learning.** Previous work in the field of optimal experiment design focused on parametric models obtained from first principles and used to determine optimal estimates of unknown parameters in their models [12, 51, 61, 65, 164, 178].
- **Proved that surprisal of the mean is a bounded approximator to Shannon entropy.** Previous work focused solely on Shannon entropy and did not consider approximations for optimal experiment design [24, 73, 164, 178].
- **Unified the information theory approach of Shannon entropy with the long-standing, intuitive metric of variance.** Variance is a popular model-based design policy [35, 51, 61, 167], which was selected historically because it was an intuitive measure. This work illustrates how weighted variance is actually the surprisal of the mean for systems with additive Gaussian noise, and provides an explanation based on information theory for why variance is a suitable design policy.
- **Demonstrated the first kinematic-based pose inference for high degree-of-freedom skeletons.** Previous work primarily relied on pose recognition for the

human skeleton [135, 154, 182]. Markerless pose estimation approaches have been limited to significantly smaller skeletons [163, 141].

Chapter 3: Modeling discrete-continuous hybrid dynamical systems

- **Presented an algorithm for the symbolic regression of piecewise functions.** The algorithm was integrated two distinct machine learning approaches: symbolic regression and expectation-maximization. Previous work on expectation-maximization for evolutionary computation focused on different optimization approaches [126, 152], as opposed to investigating different types of models found through evolutionary computation.
- **Presented an algorithm for binary classification using a symbolic regression framework.** Previous work on classification using evolutionary computation used unique representations rather than reusing a symbolic regression framework [6, 88, 130, 136].
- **Presented an algorithm for inferring symbolic models of hybrid automata models.** Previous work focused on piecewise, linear models [56, 206] or numeric approaches [9, 30, 69, 125, 109]
- **Discovered a multi-modal transistor model from unlabeled data.** To the best of my knowledge, there has been no previous work on the automated structural modeling of transistor dynamics.

Chapter 4: Uncovering hidden dynamical variables

- **Derived the necessary and sufficient conditions for state space transformations of dynamical systems.** Previous work focused on third-order systems and their transformation to a single, higher order differential equation [50].

- **Presented an algorithm for the numerical integration of black-box systems through finite differences and adaptive stepsize Runge-Kutta methods via root finding methods.** To the best of my knowledge, there has been no previous work on the numerical integration of black-box systems.
- **Presented an approach to identify non-trivial, implicit ordinary differential equations based on a principle of predictability.** Previous work on identifying implicit equations used partial derivatives [173], as opposed to a principle of predictability based on numerical solvers.
- **Presented an approach to uncover hidden dynamical variables based on a principle of parsimony.** To the best of my knowledge, there has been no previous work on uncovering meaningful hidden variables from symbolic expressions inferred from time-series data.

Chapter 5: Automated telescience

- **Presented machine discovered models of Luciferase and TCPO kinetics.** To the best of my knowledge, there has been no previous work on machine discovered models of luminescent chemical reactions.
- **Presented a new approach to combat disengagement in coevolutionary systems based on a distance metric.** Previous work dealt with disengagement by finding successively simpler problems [15].
- **Designed an infrastructure for remote operated, automated science.** Key contributions include: parameterization of experiment design, optimal experiment design, experiment design for differential equation models, and segmented spline smoothing for discontinuous systems. To the best of my knowledge, there has been no previous work on a unified approach to the automated discovery of dynamical models.

- **Presented machine discovered models of a multi-compartment microfluidic device.** To the best of my knowledge, there has been no previous work on machine discovered models of multi-compartment microfluidic devices.

Contributions of others

This section lists the effort of others who directly contributed to the results presented in this thesis. This list does not include indirect contributions through prior work that is cited in the thesis.

Chapter 2: Coevolutionary search methods and applications

- **Spider quadruped robot.** Josh Bongard, Victor Zykov and Hod Lipson [18] designed the robot that was used as the RGBD image subject.
- **RGBD images of human poses.** John R. Amend, Jr., Robert MacCurdy, and Jonas Neubert volunteered to pose for the RGBD human image set.

Chapter 3: Modeling discrete-continuous hybrid dynamical systems

- **Binary classification for symbolic regression.** Conversations with Michael D. Schmidt led to the design of this approach.
- **The testbench of hybrid systems.** Conversations with Hadas Kress-Gazit and Robert MacCurdy helped to select the set of synthetic case problems.
- **Experimental setup for voltage sweep measurements.** Jonathan Shu helped to set up the experimental infrastructure for the Keithley 2400 general purpose sourcemeter.

Chapter 4: Uncovering hidden dynamical variables

- **Numerical methods and approaches.** Conversations with Giles Hooker led to the design of many numerical methods used in this section.
- **Experimental setup for measuring Chua circuit voltages.** Robert MacCurdy helped to set up the experimental infrastructure for the Agilent DSO-X 3034 digital oscilloscope.

Chapter 5: Automated telescience

- **Literature on optimal design.** Matthew S. Shotwell provided a series of reviews on optimal experiment design for pharmacokinetic modeling of multi-compartmental systems.
- **Luminescent chemical reactions.** Christina C. Marasco and John P. Wikswo chose the luminescent chemical reactions to study as an approximation of Michaelis-Menten kinetics.
- **MATLAB Luciferase simulation model.** Christina C. Marasco designed the MATLAB Luciferase simulation model.
- **COMSOL TCPO simulation model.** Christina C. Marasco and Ryan Planchard designed the COMSOL TCPO simulation model.
- **TCPO simulation model.** Christina C. Marasco and Ryan Planchard designed the COMSOL TCPO simulation model.
- **Luminescent chemistry instrument.** Ron R. Reiserer and Christina C. Marasco designed and built the physical device.
- **Numerical methods and approaches.** Conversations with Giles Hooker led to the design of many numerical methods used in this section.

- **Experiment communication protocol and FTP server interface.** David L. McLean and Philip C. Samson designed the experiment communication protocol and the corresponding FTP server for remote operation of the pump-microscope assembly.
- **Pump-microscope-microfluidic device assembly.** David L. McLean, Philip C. Samson and John P. Wikswo designed the automated microscope device and the microfluidic chamber.

BIBLIOGRAPHY

- [1] N. Ailon. An Active Learning Algorithm for Ranking from Pairwise Preferences with an Almost Optimal Query Complexity. *Journal of Machine Learning Research*, 13:137–164, 2012.
- [2] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.
- [3] L. A. Albert and D. E. Goldberg. Efficient Discretization Scheduling in Multiple Dimensions. *Genetic and Evolutionary Computation Conference (GECCO)*, pages 271–278, 2002.
- [4] A. P. A. Anderson. A hybrid mathematical model of solid tumour invasion: the importance of cell adhesion. *Mathematical Medicine and Biology*, 22:163–186, 2005.
- [5] P. W. Anderson and E. Abrahams. Machines Fall Short of Revolutionary Science. *Science*, 324:1515–1516, 2009.
- [6] A. Arslan and M. Kaya. Determination of fuzzy logic membership functions using genetic algorithms. *Fuzzy Sets and Systems*, 118:297–306, 2001.
- [7] A. C. Atkinson and V. V. Fedorov. The design of experiments for discriminating between two rival models. *Biometrika*, 62:57–70, 1975.
- [8] Y. Baram, R. E. Yaniv, and K. Luz. Online Choice of Active Learning Algorithms. *Journal of Machine Learning Research*, 5:255–291, 2004.
- [9] Y. Bengio and P. Frasconi. An input-output HMM architecture. *Advances in Neural Information Processing Systems*, 7:427–434, 1994.
- [10] A. Berlanda, A. Sanchis, P. Isasi, and J. M. Molinda. A General Learning Co-Evolution Method to Generalize Autonomous Robot Navigation Behavior. *IEEE Congress on Evolutionary Computation (CEC)*, pages 769–776, 2000.
- [11] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [12] X. Bombois, G. Scorletti, M. Gevers, P. M. J. Van den Hof, and R. Hildebrand. Least costly identification experiment for control. *Automatica*, 42:1651–1662, 2006.

- [13] J. Bongard and H. Lipson. Automating Genetic Network Inference with Minimal Physical Experimentation Using Coevolution. *Genetic and Evolutionary Computation Conference (GECCO)*, pages 333–345, 2004.
- [14] J. Bongard and H. Lipson. Active Coevolutionary Learning of Deterministic Finite Automata. *Journal of Machine Learning Research*, 6:1651–1678, 2005.
- [15] J. Bongard and H. Lipson. ‘Managed Challenge’ Alleviates Disengagement in Co-evolutionary System Identification. *Genetic and Evolutionary Computation Conference (GECCO)*, pages 531–538, 2005.
- [16] J. Bongard and H. Lipson. Nonlinear System Identification Using Coevolution of Models and Tests. *IEEE Transactions on Evolutionary Computation*, 9:361–384, 2005.
- [17] J. Bongard and H. Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Science*, 104(24):9943–9948, 2007.
- [18] J. Bongard, V. Zykov, and H. Lipson. Resilient Machines Through Continuous Self-Modeling. *Science*, 314:1118–1121, 2006.
- [19] G. E. P. Box and W. J. Hill. Discrimination among Mechanistic Models. *Technometrics*, 9(1):57–71, 1967.
- [20] W. E. Boyce and R. C. DiPrima. *Elementary Differential Equations and Boundary Value Problems*. Wiley & Sons, Hoboken, NJ, 2005.
- [21] M. S. Branicky. *Handbook of Networked and Embedded Control Systems*, chapter “Introduction to hybrid systems”, pages 91–116. Birkhauser, 2005.
- [22] J. Branke, C. Schmidt, and H. Schmeck. Efficient fitness estimation in noisy environment. *Genetic and Evolutionary Computation Conference (GECCO)*, pages 243–250, 2001.
- [23] L. Breiman. Statistical Modeling: The Two Cultures. *Statistical Science*, 16(3):199–231, 2001.
- [24] C. H. Bryant and S. H. Muggleton. Closed Loop Machine Learning, 2000. University of York Technical Report YCS 330.

- [25] A. Bucci. *Emergent Geometric Organization and Informative Dimensions in Co-evolutionary Algorithms*. PhD thesis, Brandeis University, 2007.
- [26] B. G. Buchanan and E. A. Feigenbaum. DENDRAL and META-DENDRAL: their application dimensions. *Artificial Intelligence*, 78:5–24, 1978.
- [27] L. T. Bui, A. A. Hussein, and D. Essam. Fitness inheritance for noisy evolutionary multi-objective optimization. *Genetic and Evolutionary Computation Conference (GECCO)*, pages 779–785, 2005.
- [28] J. C. Butcher. *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods*. Wiley, New York, 1987.
- [29] C. Castellano, S. Fortunato, and V. Loreto. Statistical physics of social dynamics. *Reviews of Modern Physics*, 81:591–646, 2009.
- [30] S. Chen, S.A. Billings, and B.M Grant. Recursive hybrid algorithm for non-linear system identification using radial basis function networks. *International Journal of Control*, 55:1051–1070, 1992.
- [31] D. Clery and D. Voss. All for One and One for All. *Science*, 308:809, 2005.
- [32] D. Cliff and G. F. Miller. Tracking the red queen: measurements of adaptive progress in co-evolutionary simulations. *European Conference on Artificial Life (ECAL)*, pages 200–218, 1995.
- [33] W. G. Cochran. Experiments for Nonlinear Functions. *Journal of the American Statistical Association*, 68(344):771–781, 1973.
- [34] D. A. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15:201–221, 1992.
- [35] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4:129–145, 1996.
- [36] N. Comstock, N. Gurmen, and H. S. Fogler. http://www.engin.umich.edu/CRE/web_mod/new/glowsticks/reactions.htm.
- [37] S. Corazza, L. Mundermann, A. M. Chaudhari, T. Demattio, C. Cobelli, and T.P. Andriacchi. A markerless motion capture system to study musculoskeletal biomechanics: visual hull and simulated annealing approach. *Annals of Biomedical Engineering*, 34(6):1019–1029, 2006.

- [38] N.L. Cramer. A representation for the adaptive generation of simple sequential programs. *International Conference on Genetic Algorithms*, pages 183–187, 1985.
- [39] J. Crank and P. Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat conduction type. *Proceedings of the Cambridge Philosophical Society*, 43(1):50–67, 1947.
- [40] T.S. Cubitt, J. Eisert, and M.M. Wolf. Extracting dynamic equations from experimental data is NP hard. *Physical Review Letters*, 108, 2012.
- [41] C. De Boor. *A Practical Guide to Splines*. Springer, 2001.
- [42] E. D. De Jong and J. B. Pollack. Multi-Objective Methods for Tree Size Control. *Genetic Programming and Evolvable Machines*, 4(3):211–233, 2003.
- [43] E. D. De Jong and J. B. Pollack. Ideal Evaluation from Coevolution. *Evolutionary Computation*, pages 152–192, 2004.
- [44] M. Deisenroth, C. Rasmussen, and D. Fox. Learning to control a low-cost manipulator using data-efficient reinforcement learning. *Robotics: Science and Systems (RSS)*, 2011.
- [45] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B*, 39(1):1–38, 1977.
- [46] D. Dickmanns, J. Schmidhuber, and A. Winklhofer. Der genetische algorithmus: eine implementierung in prolog. Fortgeschrittenenpraktikum, Institut für Informatik, Technische Universität München, 1987.
- [47] R. Dillmann. Teaching and learning of robot tasks via observation of human performance. *Robotics and Autonomous Systems*, 47:109–116, 2004.
- [48] J. Doucette and M. I. Heywood. GP Classification under Imbalanced Data Sets: Active Sub-sampling and AUC approximation. *European Conference on Genetic Programming*, pages 266–277, 2009.
- [49] G. Duffing. *Erzwungene Schwingungen bei Veranderlicher Eigenfrequenz*. F. Vieweg u. Sohn, Braunschweig, 1918.

- [50] R. Eichhorn, S. J. Linz, and P. Hanggi. Transformations of nonlinear dynamical systems to jerky motion and its application to minimal chaotic flows. *Physical Review E*, 58(6):7151–7164, 1998.
- [51] A. F. Emery, A. V. Nenarokomov, and T. D. Fadale. Uncertainties in parameter estimation: the optimal experiment design. *International Journal of Heat and Mass Transfer*, 36:3331–3339, 2000.
- [52] J. Evans and A. Rzhetsky. Machine Science. *Science*, 329:399–400, 2010.
- [53] B. Falkenhainer and R. Michalski. Integrating quantitative and qualitative discovery: the ABACUS system. *Machine Learning*, 1(4):367–401, 1986.
- [54] V. Fedorov. Computational Statistics. *Optimal experiment design*, 2(5):581–589, 2010.
- [55] E. A. Feigenbaum and B. G. Buchanan. DENDRAL and META-DENDRAL: Roots of knowledge systems and expert system applications. *Artificial Intelligence*, 59:223–240, 1993.
- [56] G. Ferrari-Trecate, M. Muselli, D. Liberati, and M. Morari. A clustering technique for the identification of piecewise affine systems. *Automatica*, 39:205–217, 2003.
- [57] R. Fitzhugh. Mathematical models of threshold phenomena in the nerve membrane. *The Bulletin of Mathematical Biophysics*, 17:257–278, 1955.
- [58] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence Through Simulated Evolution*. Wiley & Sons. New York, NY., 1966.
- [59] I. Ford, D. M. Titterton, and C. P. Kitsos. Recent Advances in Nonlinear Experimental Design. *Technometrics*, 31(1):49–60, 1989.
- [60] S. Forrest. Genetic Algorithms: Principles of Natural Selection Applied to Computation. *Science*, 261:872–878, 1993.
- [61] U. Forssell and L. Ljung. Some results on optimal experiment design. *Automatica*, 36:749–756, 2000.
- [62] A. Frank and A. Asuncion. UCI Machine Learning Repository, 2010. <http://archive.ics.uci.edu/ml>.

- [63] J. Gall, C. Stoll, E. de Aguiar, C. Theobalt, B. Rosenbahn, and H.P. Seidel. Motion capture using joint skeleton tracking and surface estimation. *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1746–1753, 2009.
- [64] V. Ganapathi, C. Plagemann, D. Koller, and S. Thrun. Real time motion capture using a single time-of-flight camera. *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.
- [65] M. Gevers. Identification for Control: From the Early Achievements to the Revival of Experiment Design. *European Journal of Control*, 11:1–18, 2005.
- [66] F. Gianfelici. Machine Science: Truly Machine-Aided Science. *Science*, 330:317, 2010.
- [67] N. S. Goel, S. C. Maitra, and E. W. Montroll. On the Volterra and Other Nonlinear Models of Interacting Populations. *Reviews of Modern Physics*, 43:231276, 1971.
- [68] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
- [69] A. M. Gonzalez, A. M. S. Roque, and J. Garcia-Gonzalez. Modeling and forecasting electricity prices with input/output hidden Markov models. *IEEE Transactions on Power Systems*, 20(1):13–24, 2005.
- [70] C. C. Gordon. US Army Anthropometric Survey Database: Downsizing, Demographic Change, and Validity of the 1988 Data in 1996. Technical Report TR-07/003, US Army Natick Research Labs, 1998.
- [71] B. T. Grenfell, K. Wilson, B. F. Finkenstadt, T. N. Coulson, S. Murray, S. D. Albon, J. M. Pemberton, T. H. Clutton-Brock, and M. J. Crawley. Noise and determinism in synchronized sheep dynamics. *Nature*, 394, 1998.
- [72] J. Guckenheimer and P. Holmes. *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*. Springer Verlag. Berlin, 1983.
- [73] C. Guestrin, A. Krause, and A. Singh. Near-optimal sensor placements in Gaussian Processes. *International Conference on Machine Learning (ICML)*, 2005.
- [74] R. Haberman. *Applied Partial Differential Equations with Fourier Series and Boundary Value Problems*. Pearson Prentice Hall. Upper Saddle River, NJ, 2004.

- [75] E. Hairer, S. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer-Verlag, Berlin, 1993.
- [76] C. Haufe, K. C. Elliott, R. M. Burian, and M. A. O'Malley. Machine Science: What's Missing. *Science*, 330:317–318, 2010.
- [77] T. A. Henzinger. The theory of hybrid automata. *Symposium on Logic in Computer Science*, pages 324–335, 1996.
- [78] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
- [79] G. S. Hornby. ALPS: the age-layered population structure for reducing the problem of premature convergence. *Genetic and Evolutionary Computation Conference (GECCO)*, pages 815–822, 2006.
- [80] G. S. Hornby. Steady state ALPS for real-valued problems. *Genetic and Evolutionary Computation Conference (GECCO)*, pages 795–802, 2009.
- [81] B. G. Horne and D. R. Hush. Bounds on the complexity of recurrent neural network implementations of finite state machines. *Neural Networks*, 9(2):243–252, 1996.
- [82] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–336, 1989.
- [83] C. H. Hsieh, S. M. Glaser, A. J. Lucas, and G. Sugihara. Distinguishing random environmental fluctuations from ecological catastrophes for the North Pacific Ocean. *Nature*, 435, 2005.
- [84] W. G. Hunter and A. M. Reiner. Designs for Discriminating between Two Rival Models. *Technometrics*, 7(3):307–323, 1965.
- [85] H. Iba. Inference of differential equation models by genetic programming. *Information Sciences*, 178:4453–4468, 2008.
- [86] R. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.
- [87] H. Jacobsson. Rule extraction from recurrent neural networks: a taxonomy and review. *Neural Computation*, 17(6):1223–1263, 2006.

- [88] I. Jagielska, C. Matthews, and T. Whitfort. An investigation into the application of neural networks, fuzzy logic, genetic algorithms and rough sets to automated knowledge acquisition for classification problems. *Neurocomputing*, 24:37–54, 1999.
- [89] Y. Jin. A Comprehensive Survey of Fitness Approximation in Evolutionary Computation. *Soft Computing Journal*, 9:3–12, 2005.
- [90] Y. Jin and J. Branke. Evolutionary optimization in uncertain environments – a survey. *IEEE Transactions on Evolutionary Computation*, 9:303–317, 2005.
- [91] Y. Jin and B. Sendhoff. Reducing fitness evaluations using clustering techniques and neural network ensembles. *Genetic and Evolutionary Computation Conference (GECCO)*, pages 688–699, 2004.
- [92] B. Johanson and R. Poli. GP-music: an interactive genetic programming system for music generation with automated fitness raters. *Conference on Genetic Programming (GP)*, pages 181–186, 1998.
- [93] S. J. Julier and J. K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92:401–422, 2004.
- [94] D. Katz, Y. Pyuro, and O. Brock. Learning to manipulate articulated objects in unstructured environments using a grounded relational representation. *Robotics: Science and Systems Conference (RSS)*, 2008.
- [95] K. L. Kausrud, A. Myserud, H. Steen, J. O. Vik, E. Ostbye, B. Cazelles, E. Framstad, A. M. Eikeset, I. Myserud, T. Solhoy, and N. C. Stenseth. Linking climate change to lemming cycles. *Nature*, 456, 2008.
- [96] J. Kennedy and R. Eberhart. Particle Swarm Optimization. *IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.
- [97] H. S. Kim and S. B. Cho. An efficient genetic algorithm with less fitness evaluation by clustering. *IEEE Congress on Evolutionary Computation (CEC)*, pages 887–895, 2001.
- [98] R. D. King, J. Rowland, S. G. Oliver, M. Young, W. Aubrey, E. Byrne, M. Liakata, M. Markham, P. Pir, L. N. Soldatova, A. Sparkes, K. E. Whelan, and A. Clare. The Automation of Science. *Science*, 324:85–89, 2009.

- [99] K. Knights and B. Bryant. *Pharmacology for Health Professionals*. Elsevier, Amsterdam, 2002.
- [100] S. Knoop, S. Vacek, and R. Hillmann. Sensor Fusion for 3D Human Body Tracking with an Articulated 3D Body Model. *International Conference on Robotics and Automation (ICRA)*, 2006.
- [101] J. Kober and J. Peters. Learning motor primitives for robotics. *International Conference on Robotics and Automation (ICRA)*, pages 2112–2118, 2009.
- [102] S. Koos, J. B. Mouret, and S. Doncieux. Automatic system identification based on coevolution of models and tests. *IEEE Congress on Evolutionary Computation (CEC)*, pages 560–567, 2009.
- [103] E. B. Kosmatopoulous, M. M. Polycarpou, M. A. Christodoulou, and P. A. Ioannou. High-Order Neural Network Structures for Identification of Dynamical Systems. *IEEE Transactions on Neural Networks*, 9(2):422–431, 1995.
- [104] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press. Cambridge, MA, 1992.
- [105] D. Kulkarni and H. A. Simon. The Processes of Scientific Discovery: The Strategy of Experimentation. *Cognitive Science*, 12:139–175, 1988.
- [106] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *Journal of the ACM*, 22(2):469–475, 1975.
- [107] P. Langley. The Computational Support of Scientific Discovery. *Int. J. Human-Computer Studies*, 53:393–410, 2000.
- [108] P. Langley and H. Simon. *Scientific discovery: Computational explorations of the creative processes*. MIT Press. Cambridge, MA, 1987.
- [109] V.L. Le, G. Bloch, and F. Lauer. Reduced-size kernel models for nonlinear hybrid system identification. *IEEE Transactions on Neural Networks*, 22(12):2398–2405, 2011.
- [110] D. B. Lenat and J. S. Brown. Why AM and Eurisko appear to work. *Artificial Intelligence*, 23(3):269–294, 1984.
- [111] S. Leonelli. Machine Science: The Human Side. *Science*, 330:317, 2010.

- [112] B. Li, T. S. Lin, L. Liao, and C. Fan. Genetic algorithm based on multipopulation competitive coevolution. *IEEE Congress on Evolutionary Computation (CEC)*, pages 225–228, 2008.
- [113] K. H. Liang, X. Yao, and C. Newton. Evolutionary search of approximated n-dimensional landscape. *International Journal of Knowledge-based Intelligent Engineering Systems*, 4(3):172–183, 2000.
- [114] E. Lindberg, E. Tamaseviciute, G. Mykolaitis, S. Bumeliene, T. Pyragiene, A. Tamasevicius, and R. Kirvaitis. Autonomous Third-Order Duffing-Holmes Type Chaotic Oscillator. *European Conference on Circuit Theory and Design*, pages 663–666, 2009.
- [115] R. K. Lindsay, B. G. Buchanan, E. A. Feigenbaum, and J. Lederberg. DEN-DRAL: a case study of the first expert system for scientific hypothesis formation. *Artificial Intelligence*, 61:209–261, 1993.
- [116] L. Ljung. *System Identification: Theory for the User*. Prentice-Hall. Englewood Cliffs, NJ, 1999.
- [117] E. N. Lorenz. Deterministic nonperiodic flow. *Journal of Atmospheric Sciences*, 20(2):130–141, 1963.
- [118] J. Lunze. *Modelling, Analysis and Design of Hybrid Systems*, chapter “What is a hybrid system?”, pages 3–14. Springer, 2002.
- [119] D. L. Ly and H. Lipson. Trainer Selection Strategies for Coevolving Rank Predictors. *IEEE Congress on Evolutionary Computation (CEC)*, 2011.
- [120] D. L. Ly and H. Lipson. Co-evolutionary Predictors for Kinematic Pose Inference from RGB-D Images. *Genetic and Evolutionary Computation Conference (GECCO)*, pages 967–974, 2012.
- [121] D. L. Ly and H. Lipson. Learning Symbolic Representations of Hybrid Dynamical Systems. *Journal of Machine Learning Research*, 13:3585–3618, 2012.
- [122] D. L. Ly and H. Lipson. Optimal Experiment Design for Coevolutionary Active Learning. *IEEE Transactions on Evolutionary Computation*, 2013.
- [123] D. L. Ly, A. Saxena, and H. Lipson. Pose Estimation from a Single Depth Image for Arbitrary Kinematic Skeletons. *RGB-D Workshop at Robotics: Science and Systems Conference (RSS)*, 2011.

- [124] S. W. Mahfoud. *Niching methods for genetic algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, 1995.
- [125] S. Marcel, O. Bernier, J. E. Viallet, and D. Collobert. Hand gesture recognition using input-output hidden Markov models. *IEEE International Conference on Automatic Face and Gesture Recognition*, pages 456–461, 2000.
- [126] A.M. Martinez and J. Virtia. Learning mixture models using a genetic version of the EM algorithm. *Pattern Recognition Letters*, 21:759–769, 2000.
- [127] T. Matsumoto. A Chaotic Attractor from Chua’s Circuit. *IEEE Transactions on Circuits and Systems*, 12:1055–1058, 1984.
- [128] B. McKay, M. J. Willis, and G. W. Barton. Using a tree structured genetic algorithm to perform symbolic regression. *Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pages 487–492, 1995.
- [129] B. A. McKinney, J. E. Crowe, H. U. Voss, P. S. Crooke, N. Barney, and J. H. Moore. Hybrid grammar-based approach to nonlinear dynamical system identification from biological time series. *Physical Review E*, 73:1–7, 2006.
- [130] R. R. F. Mendes, F. B. Voznika, A. A. Freitas, and J. C. Nievola. Discovering fuzzy classification rules with genetic programming and co-evolution. *ECML Principles and Practice of Knowledge Discover in Databases*, 2168:314–325, 2001.
- [131] L. Menten and M. I. Michaelis. Die Kinetik der Invertinwirkung. *Biochemische Zeitschrift*, 49:333–369, 1913.
- [132] Microsoft Corp. Kinect for Xbox 360.
- [133] T. M. Mitchell. Mining Our Reality. *Science*, 326:1644–1645, 2009.
- [134] E. Mjolsness and D. DeCoste. Machine Learning for Science: State of the Art and Future Prospects. *Science*, 293:2051–2055, 2001.
- [135] T. Moeslund, A. Hilton, and V. Kruger. Survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104(2-3):90–126, 2006.

- [136] D. P. Muni, N. R. Pal, and J. Das. A novel approach to design classifiers using genetic programming. *IEEE Transactions on Evolutionary Computation*, 8(2):183–196, 2004.
- [137] J. Nagumo, S. Arimoto, and S. Yoshizawa. An active pulse transmission line simulating nerve axon. *Proceedings of the IRE*, 50:2061–2070, 1962.
- [138] X. H. Nguyen, R. I. McKay, and D. L. Essam. Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: the comparative results. *Australian Journal of Intelligent Information Processing Systems*, 7:114–121, 2001.
- [139] B. Nordhausen and P. Langley. A robust approach to numeric discovery. *International Conference on Machine Learning (ICML)*, pages 411–418, 1990.
- [140] K. Ogata. *Modern Control Engineering*. Pearson Prentice-Hall. Upper Saddle River, NJ, 2002.
- [141] I. Oikonomidis, N. Kyriazis, and A. A. Argyros. Efficient Model-based 3D tracking of Hand Articulations using Kinect. *British Machine Vision Conference (BMVC)*, 2009.
- [142] J. R. Olsson. Inductive functional programming using incremental program transformation. *Artificial Intelligence*, 74(1):55–81, 1995.
- [143] C. W. Omlin and C. L. Giles. Extraction, insertion and refinement of symbolic rules in dynamically driven recurrent neural networks. *Connection Science*, 5(3-4):307–337, 1993.
- [144] C. W. Omlin and C. L. Giles. Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM*, 43(6):937–972, 1996.
- [145] C. W. Omlin and C. L. Giles. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52, 1996.
- [146] Y. S. Ong, P. B. Nair, and A. J. Keane. Evolutionary optimization of computationally expensive problems via surrogate modelling. *AIAA Journal*, 41:687–696, 2003.
- [147] L. Pagie and P. Hogeweg. Evolutionary Consequences of Coevolving Targets. *Evolutionary Computation*, 5:401–418, 1997.

- [148] L. Panait and S. Luke. Cooperative Multi-Agent Learning: The State of the Art. *Autonomous Agents and Multi-Agent Systems*, pages 387–434, 2005.
- [149] S. Paoletti, A. L. Juloski, G. Ferrari-Trecate, and R. Vidal. Identification of hybrid systems: a tutorial. *European Journal of Control*, 13:242–260, 2007.
- [150] Park, J. and Sandberg, I. W. Universal approximation using radial-basis-function networks. *Neural Computation*, 3:246–257, 1991.
- [151] M. Pelikan and K. Sastry. Fitness inheritance in the Bayesian optimization algorithm. *Genetic and Evolutionary Computation Conference (GECCO)*, pages 48–59, 2004.
- [152] R. Pernkopf and D. Bouchaffra. Genetic-base EM algorithm for learning Gaussian mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1344–1348, 2005.
- [153] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published via <http://lulu.com> and freely available at <http://www.gp-field-guide.org.uk>, 2008.
- [154] R. Poppe. Vision-based human motion analysis: an overview. *Computer Vision and Image Understanding*, 108(1-2):4–18, 2007.
- [155] Press, W. H. and Teukolsky, S. A. and Vetterling, W. T. and Flannery, B.P. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, Cambridge, 2007.
- [156] I. Prigogine and R. Lefever. Symmetry Breaking Instabilities in Dissipative Systems. *Journal of Chemical Physics*, 48(4), 1968.
- [157] PyParser. <http://pyparsing.wikispaces.com>.
- [158] L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286, 1989.
- [159] H. P. Rang. *Pharmacology*. Churchill Livingstone, Edinburgh., 2003.
- [160] B. D. Reger, K. M. Fleming, V. Sanguineti, S. Alford, and F. A. Mussa-Ivaldi. Connecting brains to robots: the development of a hybrid system for the study of learning in neural tissues. *International Conference on Artificial Life*, pages 263–272, 2000.

- [161] M. Riley, A. Ude, K. Wade, and C. G. Atkeson. Enabling real-time full-body imitation: a natural way of transferring human movement to humanoids. *International Conference on Robotics and Automation (ICRA)*, pages 2368–2374, 2003.
- [162] Riordan, M. and Rowson, P.C. and Wu, S.L. The Search for the Higgs Boson. *Science*, 291:259–260, 1994.
- [163] C. Robertson and E. Trucco. Human Body Posture via Hierarchical Evolutionary Optimization. *British Machine Vision Conference (BMVC)*, 2006.
- [164] C. R. Rojas, J. S. Welsh, G. C. Goodwin, and Feuer A. Robust optimal experiment design for system identification. *Automatica*, 43:993–1008, 2007.
- [165] C. D. Rosin and R. K. Belew. New Methods for Competitive Coevolution. *Evolutionary Computation*, 5(1):1–29, 1997.
- [166] O. E. Rossler. New Methods for Competitive Coevolution. *An Equation for Continuous Chaos*, 57(5):397–398, 1976.
- [167] D. Rubens, D. Kaplan, and M. Sugiyama. Active learning in recommender systems. In *Recommender Systems Handbook*. Springer, 2011.
- [168] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.
- [169] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall. Upper Saddle River, NJ, 1995.
- [170] T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber. PyBrain. *Journal of Machine Learning Research*, 11:743–746, 2010.
- [171] M. D. Schmidt and H. Lipson. Comparison of tree and graph encodings as a function of problem complexity. *Genetic and Evolutionary Computation Conference (GECCO)*, pages 1674–1679, 2007.
- [172] M. D. Schmidt and H. Lipson. Coevolution of Fitness Predictors. *IEEE Transactions on Evolutionary Computation*, 12(6):736–749, 2008.
- [173] M. D. Schmidt and H. Lipson. Distilling Free-Form Natural Laws from Experimental Data. *Science*, 324(5923):81–85, 2009.

- [174] M. D. Schmidt and H. Lipson. Symbolic regression of implicit equations. *Genetic Programming Theory and Practice*, 7:73–85, 2009.
- [175] M. D. Schmidt and H. Lipson. Age-Fitness Pareto Optimization. *Genetic Programming Theory and Practice*, 8:129–146, 2010.
- [176] M. D. Schmidt and H. Lipson. Predicting Solution Rank to Improve Performance. *Genetic and Evolutionary Computation Conference (GECCO)*, pages 949–956, 2010.
- [177] M. D. Schmidt and H. Lipson. Eureqa, 2013. <http://creativemachines.cornell.edu/eureqa>.
- [178] P. Sebastiani and H. P. Wynn. Maximum entropy sampling and optimal Bayesian experimental design. *Journal of the Royal Statistical Society B*, 62:145–157, 2000.
- [179] A. S. Sedra and K. C. Smith. *Microelectronic Circuits*. Oxford University Press. Oxford, UK, 2004.
- [180] B. Settles. Active Learning Literature Survey, 2010. University of Wisconsin-Madison, Computer Sciences Technical Report 1648.
- [181] K. Shoemake. Animating rotation with quaternion curves. *ACM SIGGRAPH Computer Graphics*, 19(3), 1985.
- [182] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finnochio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from a single depth image. *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.
- [183] L. Sigal, S. Bhatia, S. Roth, M.J. Black, and M. Isard. Tracking loose-limbed people. *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.
- [184] S. Simic. Best possible global bounds for Jensen’s inequality. *Applied Mathematics and Computation*, pages 2224–2228, 2009.
- [185] R. E. Smith, B. A. Dike, and S. A. Stegmann. Fitness inheritance in genetic algorithms. *ACM Symposium on Applied Computing*, pages 345–350, 1995.

- [186] S. F. Smith. *A Learning System Based on Genetic Adaptive Algorithms*. PhD thesis, University of Pittsburgh, 1980.
- [187] R.J. Solomonoff. A formal theory of inductive inference I, II. *Information and Control*, 7:1–22, 224–254, 1964.
- [188] D. Song, M. I. Heywood, and A. N. Zincir-Heywood. Training Genetic Programming on Half a Million Patterns: An Example from Anomaly Detection. *IEEE Transactions on Evolutionary Computation*, 9(3):225–240, 2005.
- [189] A. Sparkes, W. Aubrey, E. Byrne, A. Clare, M. N. Khan, M. Liakata, M. Markham, J. Rowland, L. N. Soldatova, K. E. Whelan, M. Young, and R. D. King. Toward Robot Scientists for Autonomous Scientific Discovery. *Automated Experimentation*, 2(1):1–11, 2010.
- [190] J. C. Sprott. Simple chaotic systems and circuits. *American Journal of Physics*, 68(8):758–763, 2000.
- [191] J. Stewart. *Calculus*. Thomson. Belmont, CA, 2003.
- [192] S.H. Strogatz. *Nonlinear Dynamics and Chaos*. Westview Press, Cambridge, MA, 1994.
- [193] G. Sugihara and R. M. May. Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series. *Nature*, 344, 1990.
- [194] J. Sung, C. Ponce, B. Selman, and A. Saxena. Unstructured Human Activity Detection from RGBD Images. *International Conference on Robotics and Automation (ICRA)*, 2012.
- [195] SymPy. <http://sympy.org>.
- [196] A. Szalay and J. Gray. Science in an exponential world. *Nature*, 440:413–414, 2006.
- [197] F. Takens. Detecting strange attractors in turbulence. *Lecture Notes in Mathematics*, 989:366–381, 1981.
- [198] G. Taylor, L. Sigal, D. Fleet, and G. E. Hinton. Dynamic binary latent variable models for 3d human pose tracking. *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

- [199] A. Teller and D. Andre. Automatically Choosing the Number of Fitness Cases: The Rational Allocation of Trials. *Conference on Genetic Programming (GP)*, pages 321–328, 1997.
- [200] C. Tomlin, G. J. Pappas, and S. S. Sastry. Conflict resolution for air traffic management: a study in multi-agent hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):509–521, 1998.
- [201] D. Ucinski and B. Bogacka. T-optimum designs for discrimination between two multiresponse dynamic models. *Journal of the Royal Statistical Society: Series B*, 67:3–18, 2005.
- [202] A. Vahed and C. W. Omlin. A machine learning method for extracting symbolic knowledge from recurrent neural networks. *Neural Computation*, 16:59–71, 2004.
- [203] B. Van der Pol. On relaxation-oscillations. *Philosophical Magazine & Journal of Science*, 2(7):978–992, 1927.
- [204] B. Van der Pol and J. Van der Mark. Frequency demultiplication. *Nature*, 120:363–364, 1927.
- [205] A. van der Schaft and H. Schumacher. *An Introduction to Hybrid Dynamical Systems*. Springer, 2000.
- [206] R. Vidal, S. Soatto, Y. Ma, and S. Sastry. An algebraic geometric approach to the identification of a class of linear hybrid systems. *Conference on Decision and Control*, pages 167–172, 2003.
- [207] A. Visintin. *Differential Models of Hysteresis*. Springer, 1995.
- [208] E. Vladislavleva, G. Smits, and D. den Hertog. Order of nonlinearity as a complexity measure for models generated by symbolic regression via Pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349, 2009.
- [209] V. Volterra. Fluctuations in the Abundance of a Species Considered Mathematically. *Nature*, 118:558–560, 1926.
- [210] C. S. Wallace and D. L. Dowe. Minimum message length and Kolmogorov complexity. *The Computer Journal*, 42(4):270–283, 1999.

- [211] D. Waltz and B. G. Buchanan. Automating Science. *Science*, 324:43–44, 2009.
- [212] M. P. Wand and M. C. Jones. *Kernel Smoothing*. Chapman and Hall. London, 1995.
- [213] R. A. Watson and J. B. Pollack. Coevolutionary dynamics in a minimal substrate. *Genetic and Evolutionary Computation Conference*, pages 702–709, 2001.
- [214] I. C. Yeh. Modeling of strength of high performance concrete using artificial neural networks. *Cement and Concrete Research*, 28(12):1797–1808, 1998.
- [215] I. C. Yeh. Analysis of strength of concrete using design of experiments and neural networks. *Journal of Materials in Civil Engineering*, 18(4):597–604, 2006.
- [216] G. Q. Zhong. Implementation of Chua’s circuit with a cubic nonlinearity. *IEEE Transactions on Circuits and Systems*, 41:934–941, 1994.
- [217] J. Zytkow. Automated discovery of empirical laws. *Fundamenta Informaticae*, 27(2):299–318, 1996.